Information and Computation 210 (2012) 68–86

Contents lists available at SciVerse ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Improved model checking of hierarchical systems ☆

Benjamin Aminof [a,1], Orna Kupferman [a], Aniello Murano [b,*,2]

[a] Hebrew University, School of Eng. and Computer Science, Jerusalem 91904, Israel
[b] Università degli Studi di Napoli "Federico II", 80126 Napoli, Italy

ARTICLE INFO

Article history:
Received 2 March 2010
Revised 4 October 2011
Available online 28 October 2011

Keywords:
Hierarchical systems
Model checking
Branching-time temporal logics
Two-player games
Parity games
Abstraction-refinement

ABSTRACT

We present a unified game-based approach for branching-time model checking of hierarchical systems. Such systems are exponentially more succinct than standard state-transition graphs, as repeated sub-systems are described only once. Early work on model checking of hierarchical systems shows that one can do better than a naive algorithm that "flattens" the system and removes the hierarchy.

Given a hierarchical system $\mathcal{S}$ and a branching-time specification ψ for it, we reduce the model-checking problem (does $\mathcal{S}$ satisfy ψ?) to the problem of solving a *hierarchical game* obtained by taking the product of $\mathcal{S}$ with an alternating tree automaton $\mathcal{A}_{\psi}$ for ψ. Our approach leads to clean, uniform, and improved model-checking algorithms for a variety of branching-time temporal logics. In particular, by improving the algorithm for solving hierarchical parity games, we are able to solve the model-checking problem for the μ-calculus in PSPACE and time complexity that is only polynomial in the depth of the hierarchy. Our approach also leads to an abstraction-refinement paradigm for hierarchical systems. The abstraction maintains the hierarchy, and is obtained by merging both states and sub-systems into abstract states.

1. Introduction

In model checking, we verify that a system meets its specification by translating the system to a finite state machine (FSM), translating the specification to a temporal-logic formula, and checking that the FSM satisfies the formula [10]. The translation of a high-level description of a system to an FSM involves a painful blow-up, and the size of the FSM is typically the computational bottleneck in model-checking algorithms.

There are several sources of the blow-up that the translation involves. A well-studied source is the ability of components in the system to work in parallel and communicate with each other, possibly using variables. Formally, *concurrent FSMs* are exponentially more succinct than *flat* (usual) ones [14]. This has led to extensive research on compositional model checking, where the goal is to reason about a system by reasoning about its underlying components and without constructing an equivalent flat system (cf., [13,28]). Compositionality methods are successfully applied in practice (cf., [29]), but it is a known reality that they cannot always work. Formally, the system complexity of the model-checking problem (that is, the complexity in terms of the system, assuming a specification of a fixed length) for all common temporal logics is

☆ A preliminary version appeared in the proceedings of VMCAI 2010, volume 5944 of Lecture Notes in Computer Science, Springer.

* Corresponding author.
E-mail addresses: benj@cs.huji.ac.il (B. Aminof), orna@cs.huji.ac.il (O. Kupferman), murano@na.infn.it (A. Murano).

[1] This work was partially done while the author was visiting Università degli Studi di Napoli "Federico II", supported by ESF GAMES project, short visit grant n.2789.

[2] Partially supported by ESF GAMES project, Vigevani Project Prize 2010–2011, and by University of Napoli Federico II under the F.A.R.O. project.

0890-5401/$ – see front matter
doi:10.1016/j.ic.2011.10.008

exponentially higher in the concurrent setting [23]. This exponential gap is carried over to other related problems such as checking language-containment and bisimulation—all are exponentially harder in the concurrent setting [20,30].

Another source of the blow-up in the translation of systems to FSMs has to do with the ability of a high-level description of a system to reuse the same component in different contexts (say, by calling a procedure). The sequential setting is that of *hierarchical FSMs*, where some of the states of the FSM are boxes, which correspond to nested FSMs. The naive approach to model checking such systems is to "flatten" them by repeatedly substituting references to sub-structures with copies of these sub-structures. However, this results in a flat system that is exponential in the nesting depth of the hierarchical system. In [6], Alur and Yannakakis show that for LTL model checking, one can avoid this blow-up altogether, whereas for CTL, one can trade it for an exponential blow-up in the (often much smaller) size of the formula and the maximal number of exits of sub-structures. In other words, while hierarchical FSMs are exponentially more succinct than flat FSMs [5], in many cases the system complexity of the model-checking problem is not exponentially higher in the hierarchical setting! Thus, even more than with the feature of concurrency, here there is clear motivation not to flatten the FSM before model checking it.

The results in [6] set the stage to further work on model-checking of hierarchical systems. As it so happened, however, this line of research has quickly been focused on *recursive systems*, which allow unbounded nesting of components. Having no bound on the nesting gives rise to infinite-state systems. The emergence of software model checking, the natural association of reusability with (possibly recursive) procedure calls, the challenge and abstraction that the infinite-state setting involves, and the neat connection to pushdown automata, have all put recursive systems in the central stage [1,2,4], leaving the hierarchical setting as a special case. This work hopes to shift some attention back to the hierarchical setting. We suggest a uniform game-based approach for model checking such systems, and argue that the game-based approach enjoys the versatility and advantages it has proven to have in the flat setting. In particular, the game-based approach leads to improved model-checking algorithms and to an abstraction-refinement framework for hierarchical systems and CTL formulas. An important conclusion of our work is that we should not hurry to give up the finite-state nature of the hierarchical setting, as it does lead to simpler algorithms, and better complexities than the recursive setting.

In the flat setting, the *game-based* approach reduces the model-checking problem (does a system $\mathcal{S}$ satisfy a branching temporal logic specification ψ?) to the problem of deciding a *two-player game* obtained by taking the product of $\mathcal{S}$ with an alternating tree automaton $\mathcal{A}_\psi$ for ψ [23]. The game-based approach separates the logic-related aspects of the model-checking problem, which are handled in the translation of the specifications to automata, and the combinatorial aspects, which are handled by the game-solving algorithm. Using the game-based approach, it was possible to tighten the time and space complexity of the branching-time model-checking problem [23]. We describe a unified game-based approach for branching-time model checking of hierarchical systems. We define *two-player hierarchical games*, and reduce model checking to deciding such games. In a hierarchical game, an arena may have boxes, which refer to nested sub-arenas. As in the flat setting, one can take the product of a hierarchical system with an alternating tree automaton for its specification, and model checking is reduced to solving the game obtained by taking this product. Now, however, the hierarchy of the system induces hierarchy in the game.

Having introduced the framework, we turn to the two main technical contributions of the paper: a new and improved algorithm for solving hierarchical parity games, and an abstraction-refinement paradigm for hierarchical systems. We now briefly describe both. Consider a hierarchical game $\mathcal{G}$. The idea behind our algorithm is that even though a sub-arena may appear in different contexts, it is possible to extract information about the sub-arena that is independent of the context in which it appears. Formally, for each strategy of one of the players, we can analyze the sub-arena and extract a *summary function*, mapping each exit of the sub-arena to the best color (of the parity condition) that the other player can hope for, given that the current play eventually leaves the sub-arena through this exit. The summary function is independent of the context and has to be calculated only once. The algorithm for solving the game $\mathcal{G}$ then solves a sequence of flat parity games, obtained by replacing sub-arenas by simple gadgets that implement the summary functions.

The idea of summarizing segments of a play in a parity game is very natural and has already been used in similar contexts. We mention here two. In [34], Walukiewicz studied parity games over pushdown systems. There, the transitions of a pushdown automaton that is constructed in order to calculate a winning strategy depend on the stack of the pushdown game. Rather than remembering the whole stack, the automaton remembers at each point in time only a summary that maps each color c to the set of control states that can be reached when the current top of stack z is popped for the first time, provided that the smallest color encountered between pushing and popping z is c. An elaboration of this idea is used in [3], which combines the pushdown system with a visibly pushdown specification. There, the variables of the specification do not range over states but rather over summaries defined for each program block.

While hierarchical systems may be exponentially more succinct than flat ones, they are not immune to the "state-explosion problem", which, in some circumstances, could completely absorb the flavor of using hierarchical state machines. For flat systems, a powerful solution to the state-explosion problem is based on reasoning about an abstraction of the concrete model. In order to guarantee preservation of the branching-time specification from abstract models to concrete models, two transition relations have been considered [12,25]: preservation of universal properties requires an over-approximation, whereas preservation of existential properties requires an under-approximation. This is accomplished by using modal transition systems [17,21]. We extend this approach to hierarchical state machines and introduce *hierarchical modal transition systems* (HMTS, for short) and *hierarchical* 3*-valued games*. We show how to abstract a hierarchical system and get an HMTS, and how to model check specifications in CTL. The abstraction technique fits into our game-based approach very naturally.

Indeed, already in the flat setting, reasoning about abstractions has the flavor of solving games [31]. From a technical point of view, combining our algorithm for the concrete hierarchical setting and the abstraction-refinement solution for the flat setting [31], is not difficult, and is based on adding to the gadgets that capture the summary functions a layer in which the players can chose between winning and not losing (i.e., forcing the game to an unknown-winner value). We see this as a witness to the neatness of our framework.

Related work As described above, the formulation of hierarchical systems as well as the observation that model-checking algorithms for them should not flatten the system, was done in [6]. The work since then was focused on recursive systems, with the exception of [18,24,27]. The closest to our work here is [18], which proved that the model-checking problem for the μ-calculus and hierarchical systems is PSPACE-complete (as opposed to the recursive setting, in which μ-calculus model checking is EXPTIME-complete). As we specify below, the μ-calculus model-checking algorithm that our approach induces enjoys several advantages with respect to the one in [18].

The first advantage is the complexity. While the algorithm in [18] is better than the naive "flattening" approach in terms of space complexity, no attention is given to its time complexity. We found no specific analysis of the time complexity of the algorithm in [18]. According to our analysis, its time complexity is always worse than the "flattening" approach. Indeed, while the "flattening" approach for model-checking a μ-calculus formula φ in a hierarchical system $\mathcal{K}$ is exponential only in the nesting depth of $\mathcal{K}$ and the alternation depth l of φ, the algorithm in [18] is super-exponential *also* in the formula and in an expression[3] that depends on the number of boxes and exits in sub-structures of $\mathcal{K}$. On the other hand, our approach, which also gives an algorithm in PSPACE, yields an algorithm with a much better time complexity of $(|\mathcal{K}| \cdot |\varphi|)^l \cdot 2^{O(|\varphi|) \cdot e \cdot l \cdot \log l}$, where e is the maximal number of exits in a sub-structure of $\mathcal{K}$. We note that in many designs e is very small (often, e is constant). Note that our algorithm is not exponential in the number of boxes in a sub-structure of $\mathcal{K}$, or in the nesting depth (the nesting-depth factor is subsumed in $|\mathcal{K}|$, in which our algorithm is polynomial). Hence, beyond having a polynomial space complexity, the time complexity of our algorithm is usually much better than the one that follows the "flattening" approach, and in all cases it is much better than the one in [18].

Second, recall that we reduce model-checking to solving hierarchical games. In particular, μ-calculus model checking is reduced to solving parity games. Our algorithm for the latter is based on solving a sequence of (non-hierarchical) parity games. As such, it can benefit from existing and future algorithms and tools for solving parity games. This has both practical and theoretical advantages. For example, while it is an easy consequence of our algorithm that hierarchical parity games over arenas with a constant number of exits can be solved by solving a polynomial number of parity games, the work in [18] had to provide a special analysis in order to show the weaker result that such games are in NP $\cap$ co-NP.

Third, the algorithm presented in [18] does not deal directly with hierarchical systems. Rather, it considers *straight line programs* (SLP) generated by a grammar with five graph rewriting rules. Translating a hierarchical system to an SLP is not hard, but it involves an application of quadratically many rules. Beyond the blow-up that such a translation involves, it messes-up the direct relationship between the structure of the hierarchical system and the game. This direct relationship is crucial in understanding the output of the model-checking procedure, by means of counterexamples or certificates, and in describing an abstraction-refinement paradigm on top of the game.

Finally, unlike the uniform treatment that our approach suggests, the algorithm presented in [18] cannot be easily generalized to handle more settings. The uniformity of our approach is reflected both in the fact that it can optimally handle many logics, and in the fact that it leads to tight complexity bounds even when we focus on different components of the model-checking problem. For example, while it is immediate from our algorithm that the model-checking problem of constant size μ-calculus formulas over hierarchical systems with a constant number of exits is in PTIME, proving the same result in [18] required arguments that are orthogonal to the algorithm there, and are based on Courcelle's technique for evaluating fixed MSO-formulas over bounded-width graphs.

2. Preliminaries

2.1. Hierarchical games and systems

A *hierarchical two-player game* is a game played between two players, referred to as Player 0 and Player 1. The game is defined by means of a hierarchical arena and a winning condition. The players move a token along the hierarchical arena, and the winning condition specifies the objectives of the players, which typically refer to the sequence of states traversed by the token. A *hierarchical arena* is a hierarchical FSM in which the state space of each of the underlying FSMs is partitioned into states belonging to Player 0 (that is, when the token is in these states, then Player 0 chooses a successor to which he moves the token) and states belonging to Player 1. We refer to the underlying FSMs as *sub-arenas*. Formally, a hierarchical two-player game is a pair $\mathcal{G} = (\mathcal{V}, \Gamma)$, where $\mathcal{V} = \langle \mathcal{V}_1, \ldots, \mathcal{V}_n \rangle$ is a hierarchical arena, and Γ is a winning condition. For every $1 \leqslant i \leqslant n$, the sub-arena $\mathcal{V}_i = \langle W_i^0, W_i^1, \mathcal{B}_i, in_i, exit_i, \tau_i, \mathcal{R}_i \rangle$ has the following elements:

[3] More specifically, it is exponential in $(w \cdot |\varphi|)^2$, where w is the maximal *calls width* of sub-structures of $\mathcal{K}$, defined by $\max_i\{\sum_{b \in B_i}(|exit_{\tau_i(b)}|)\}$. Note that while the number of exits $|exit_{\tau_i(b)}|$, in the sub-structure that a box b refers to, is usually small; the number of boxes $|B_i|$ can be very big.

- W_i^0 and W_i^1 are finite sets of *states*. States in W_i^0 belong to Player 0, and states in W_i^1 belong to Player 1. We assume that $W_i^0 \cap W_i^1 = \emptyset$, and let $W_i = W_i^0 \cup W_i^1$. The state $in_i \in W_i$ is an *initial state*,[4] and $exit_i \subseteq W_i$ is a set of *exit-states*. We assume that $exit_1 = \emptyset$, i.e., the top-level arena $\mathcal{V}_1$ has no exits.
- A finite set $\mathcal{B}_i$ of *boxes*. We assume that $W_1, \ldots, W_n, \mathcal{B}_1, \ldots, \mathcal{B}_n$ are pairwise disjoint.
- An indexing function $\tau_i : \mathcal{B}_i \to \{i+1, \ldots, n\}$ that maps each box of the i-th sub-arena to an index greater than i. If $\tau_i(b) = j$, we say that b *refers* to $\mathcal{V}_j$.
- An edge relation $\mathcal{R}_i \subseteq (\bigcup_{b \in \mathcal{B}_i} (\{b\} \times exit_{\tau_i(b)}) \cup W_i) \times (W_i \cup \mathcal{B}_i)$. Let the pair (u, v) be an edge in $\mathcal{R}_i$, with a source u and a target v. The source u is either a state of $\mathcal{V}_i$ or a pair (b, e), where b is a box of $\mathcal{V}_i$ and e is an exit-state of the sub-arenas that b refers to. The target v is either a state or a box of $\mathcal{V}_i$.

In a sub-arena, the edges connect states and boxes with one another. Edges entering a box implicitly lead to the unique initial state of the sub-arena that the box refers to. On the other hand, an edge exiting a box explicitly specifies the exit-state it comes out of. Note that the fact that boxes can refer only to sub-arenas of a greater index implies that the nesting depth of arenas is finite. In contrast, in the *recursive* setting such a restriction does not exist [1].

A parity winning condition Γ for the game maps all states (of all sub-arenas) to a finite set of colors $C = \{C_{\min}, \ldots, C_{\max}\} \subset \mathbb{N}$. Thus, $\Gamma : \bigcup_i W_i \to C$. For technical convenience we allow Γ to be partial, but require that in every sub-arena every cycle, as well as every path from an entry to an exit, has at least one colored state.

A *hierarchical structure* (*hierarchical system*) can be viewed as a hierarchical arena with a single player. In addition, the structure is defined with respect to a set *AP* of *atomic propositions*, and each state of the structure is mapped to the set of propositions that hold in it. Formally, a hierarchical structure over *AP* is a tuple $\mathcal{K} = \langle \mathcal{K}_1, \ldots, \mathcal{K}_n \rangle$ of *structures*, where each $\mathcal{K}_i = \langle AP, \mathcal{V}_i, \sigma_i \rangle$ has a sub-arena $\mathcal{V}_i$ with $W_i^1 = \emptyset$, and a labeling function $\sigma_i : W_i \times AP \to \{tt, ff\}$ that assigns a truth value to a pair $(w, p) \in W_i \times AP$, which indicates whether the atomic proposition p holds or not in w. For convenience, we sometimes abuse notation and write $\sigma_i(w)$ to denote the set $\{p \in AP : \sigma_i(w, p) = tt\}$.

A sub-arena without boxes is *flat*. A flat sub-arena that has no exits is *simple*. A game over a flat (resp. simple) arena is called a flat (resp. simple) game. The special case of a simple hierarchical structure is the classical Kripke structure. Each hierarchical arena $\mathcal{V}$ can be transformed to an equivalent flat arena $\mathcal{V}^f$ (called its *flat expansion*) by recursively substituting each box by a copy of the sub-arena it refers to. Since different boxes can refer to the same sub-arena, states may appear in different contexts. In order to obtain unique names for states in the flat arena, we prefix each copy of a sub-arena's state by the sequence of boxes through which it was reached. Thus, a state $(b_0, \ldots, b_k, w)$ of $\mathcal{V}^f$ is a vector whose last component w is a state of $\mathcal{V}$, and the remaining components $(b_0, \ldots, b_k)$ are boxes that describe its context. For simplicity, we refer to vectors of length one as elements (that is, w, rather than (w)).

Formally, given a hierarchical arena $\mathcal{V} = \langle \mathcal{V}_1, \ldots, \mathcal{V}_n \rangle$, for each sub-arena $\mathcal{V}_i$ we inductively define its flat expansion $\mathcal{V}_i^f = \langle W_i^{0^f}, W_i^{1^f}, \emptyset, in_i, exit_i, \emptyset, \mathcal{R}_i^f \rangle$ as follows.[5]

- For $\sigma \in \{0, 1\}$, the set $W_i^{\sigma f} \subseteq W_i^\sigma \cup (\mathcal{B}_i \times (\bigcup_{j=i+1}^n W_j^{\sigma f}))$ is defined as follows:
  - If w is a state of W_i^σ, then w belongs to $W_i^{\sigma f}$.
  - If b is a box of $\mathcal{V}_i$ with $\tau_i(b) = j$, and the tuple $(u_1, \ldots, u_h)$ is a state in $W_j^{\sigma f}$, then $(b, u_1, \ldots, u_h)$ belongs to $W_i^{\sigma f}$.
- The transition relation $\mathcal{R}_i^f$ is defined as follows:
  - If $(u, v) \in \mathcal{R}_i$, where $u \in W_i$ or $u = (b, e)$, where $b \in \mathcal{B}_i$ and $e \in exit_{\tau_i(b)}$, then if the target v is a state then $(u, v) \in \mathcal{R}_i^f$; and if v is a box then $(u, (v, in_{\tau_i(v)})) \in \mathcal{R}_i^f$. Note that $(v, in_{\tau_i(v)})$ is indeed a state of W_i^f by the second item in the definition of states above.
  - If b is a box of $\mathcal{V}_i$, and $((u_1, \ldots, u_h), (v_1, \ldots, v_{h'}))$ is a transition of $\mathcal{V}_{\tau_i(b)}^f$, then $((b, u_1, \ldots, u_h), (b, v_1, \ldots, v_{h'}))$ belongs to $\mathcal{R}_i^f$.

The arena $\mathcal{V}_1^f$ is the required flat expansion $\mathcal{V}^f$ of $\mathcal{V}$. Let $W_i^f = W_i^{0^f} \cup W_i^{1^f}$. In case $\mathcal{K} = \langle \mathcal{K}_1, \ldots, \mathcal{K}_n \rangle$ is a hierarchical structure, where each $\mathcal{K}_i = \langle AP, \mathcal{V}_i, \sigma_i \rangle$ is a structure over *AP*, then the flat expansion is $\mathcal{K}_i^f = \langle AP, \mathcal{V}_i^f, \sigma_i^f \rangle$, where the labels are induced by the innermost state. Thus, $\sigma_i^f : W_i^f \times AP \to \{tt, ff\}$ is such that for every $p \in AP$, if $w = (u_1, \ldots, u_h)$, then $\sigma_i^f(w, p) = \sigma_j(u_h, p)$, where j is the index of the structure of which u_h is a state of. A hierarchical structure $\mathcal{K}$ satisfies a formula φ (denoted $\mathcal{K} \models \varphi$) iff its flat expansion $\mathcal{K}^f$ does. The *hierarchical model-checking problem* is to decide, given a hierarchical structure $\mathcal{K}$ and temporal-logic formula φ, whether $\mathcal{K}$ satisfies φ.

The semantics of a game over a hierarchical arena is defined by means of its flat expansion, and thus the definitions of a play, a strategy, etc. are essentially the classic definitions for flat games. However, for our purpose, it is convenient to also consider plays over arenas $\mathcal{V}_i$, for $1 < i \leqslant n$, which are not the top-level arena $\mathcal{V}_1$. Such arenas may have exit nodes, and we adjust the definitions to deal with these exits. Intuitively, a play of a game over $\mathcal{V}_i$ proceeds by moving a token on

[4] We assume a single entry for each sub-arena. Multiple entries can be handled by duplicating sub-arenas.

[5] We note that, unlike the definition of flat structures in [6], our definition of flat arenas also refers to exits. This is useful in the solution of games.

the nodes of the flat expansion $\mathcal{V}_i^{\mathrm{f}}$, starting at the initial node in_i. If the token is placed on a node $s \in W_i^{0\,\mathrm{f}}$ then Player 0 chooses the next move, and if it is placed on a node $s \in W_i^{1\,\mathrm{f}}$ then Player 1 is doing the choosing. The available moves are as follows. If s has no successors in $\mathcal{V}_i^{\mathrm{f}}$, and $s \notin exit_i$ (we call such a node a *terminal node*), then the play ends; Otherwise, the player chooses a successor of s and moves the token to this successor, or, if $s \in exit_i$, he may choose instead to move the token "outside" $\mathcal{V}_i^{\mathrm{f}}$, in which case the play also ends. A *play* of the game is thus a (finite or infinite) sequence of nodes $\pi = \pi_0, \pi_1, \ldots$, namely, the sequence of nodes the token has traversed during the play, with possibly the symbol *out* at the end of a finite sequence (indicating that the token was moved out of the arena). A play π is *initial* if $\pi_0 = in_i$; it is *maximal* if it is (i) initial, and (ii) it is infinite, or it is finite but it cannot be extended to a longer play. Note that we sometimes refer to plays as words in $(W^{\mathrm{f}})^\omega + (W^{\mathrm{f}})^* + (W^{\mathrm{f}})^* \cdot \{out\}$.

Consider a parity winning condition Γ. For a play π, let $maxC(\pi)$ be the maximal color that appears infinitely often along π (recall that by our assumptions an infinite play must have infinitely many colored nodes), or appears at least once if π is finite and has at least one colored node. A play is winning for Player 0 if it ends in a terminal node $s \in W_i^{1\,\mathrm{f}}$, i.e., if Player 1 cannot extend the play; or if the play is infinite and satisfies Γ, i.e., $maxC(\pi)$ is even. Similarly, a play is winning for Player 1 if it ends in a terminal node $s \in W_i^{0\,\mathrm{f}}$, or if the play is infinite and does not satisfy the winning condition Γ. A play that ends with *out* (i.e., because the token was moved outside the arena) is not winning for either player, and has an undefined value.

A *strategy* for a player is a function from prefixes of plays ending in one of his nodes, to the set of nodes plus the action *out*, telling Player σ what move to make in order to extend the play. Thus, for $\sigma \in \{0,1\}$, a Player σ strategy is a partial function $\xi : (W^{\mathrm{f}})^* \cdot W_i^{\sigma\,\mathrm{f}} \to (W^{\mathrm{f}} \cup \{out\})$, such that for all $u \cdot v$, with $u \in (W^{\mathrm{f}})^*$ and $v \in W_i^{\sigma\,\mathrm{f}}$, we have that $\xi(u \cdot v) = out$ only if $v \in exit_i^{\mathrm{f}}$, and otherwise, $(v, \xi(u \cdot v)) \in \mathcal{R}_i^{\mathrm{f}}$. A prefix $\pi_0, \ldots, \pi_n$ is consistent with a strategy ξ of Player σ, if for all $j \geqslant 0$ it holds that if π_j is a Player σ node then $\pi_{j+1} = \xi(\pi_0, \ldots, \pi_j)$. The function is partial as there may be vertices in $W_i^{\sigma\,\mathrm{f}}$ with no successors, and since we do not require it to be defined over plays that are not consistent with it. A strategy ξ is *memoryless* if its output does not depend on the whole prefix of the play, but only on the last position, i.e., if for all $u, u' \in (W^{\mathrm{f}})^*$ and all $v \in W_i^{\sigma\,\mathrm{f}}$, we have that $\xi(u \cdot v) = \xi(u' \cdot v)$. We can thus abbreviate and think of a memoryless strategy for Player σ as a partial function $\xi : W_i^{\sigma\,\mathrm{f}} \to (W^{\mathrm{f}} \cup \{out\})$. Observe that if $b_1, b_2 \in \mathcal{B}_i$ are two boxes that refer to the same sub-arena $\mathcal{V}_j$, then it is normally *not* the case that ξ (even if it is memoryless) behaves in the same way, inside $\mathcal{V}_j$, in both cases. That is, the choice of how to move inside $\mathcal{V}_j$ depends on the context in which it appears.

It is easy to see that for every two strategies, ξ^0 for Player 0 and ξ^1 for Player 1, there is exactly one play consistent with both strategies. Thus, two strategies induce a play. We denote this play by $outcome(\xi^0, \xi^1)$. A strategy ξ^σ for Player σ is *winning*, if for all strategies $\xi^{1-\sigma}$ for Player $1-\sigma$, the play $outcome(\xi^0, \xi^1)$ is winning for Player σ. Dually, a strategy ξ^σ for Player σ is *losing*, if there exists a strategy $\xi^{1-\sigma}$ for Player $1-\sigma$, for which the play $outcome(\xi^0, \xi^1)$ is winning for Player $1-\sigma$. Note that since plays that end with *out* have an undefined value, a strategy ξ^σ may be neither winning nor losing. Also note that if ξ^σ is not a losing strategy for Player σ, then all plays agreeing with ξ^σ that do not end with *out* are winning for Player σ. If the arena $\mathcal{V}_i$ has no exits, i.e., if $exit_i = \emptyset$, then neither does $\mathcal{V}_i^{\mathrm{f}}$, and the semantics of a game over $\mathcal{V}_i$ coincides with the classic definition for parity games over simple arenas. By [15], parity games are *determined* with memoryless strategies over simple arenas, i.e., it is always the case that one of the players (called the *winner* of the game) has a memoryless winning strategy. To *solve* a game over an arena with no exits is to find the winner of the game.

Observe that an alternative way of looking at the semantics of a game over the hierarchical arena $\mathcal{V}_i$ is to think of the token as being moved directly on the nodes of the sub-arenas $\mathcal{V}_i, \ldots, \mathcal{V}_n$, using an auxiliary stack to keep track of the context. Recall that a node $s = (b_0, \ldots, b_k, w)$ of $\mathcal{V}_i^{\mathrm{f}}$ is a vector whose last component w is a node in $\bigcup_{j=i}^n (W_j)$, and the remaining components $b_0, \ldots, b_k$ are boxes in $\bigcup_{j=i}^n (\mathcal{B}_j)$ that give its context. Thus, a token that is positioned on s can be represented by a token positioned on w, with an auxiliary stack containing $b_1 \cdots b_k$. Since the arena is hierarchical (and not recursive) the depth of the stack is bounded.

The *size* $|\mathcal{V}_i|$ of a sub-arena $\mathcal{V}_i$ is the sum $|W_i| + |\mathcal{B}_i| + |\mathcal{R}_i|$, and the number of exits of $\mathcal{V}_i$ is $|exit_i|$. The size $|\mathcal{V}|$ of a hierarchical arena $\mathcal{V}$ is the sum of the sizes of all its sub-arenas $\mathcal{V}_i$, and the number of its exits $exits(\mathcal{V}) = \max_i(|exit_i|)$ is the maximal number of exits in any of its sub-arenas. The nesting depth of $\mathcal{V}$, denoted $nd(\mathcal{V})$, is the length of the longest chain $i_1, i_2, \ldots, i_j$ of indices such that a box of $\mathcal{V}_{i_l}$ is mapped to i_{l+1}. Observe that each state of the expanded structure is a vector of length at most the nesting depth, and that the size of $\mathcal{V}^{\mathrm{f}}$ can be exponential in the nesting depth, i.e., $\Omega(|\mathcal{V}|^{nd(\mathcal{V})})$.

2.2. Alternating parity tree automata

Let $\mathcal{D}$ be a set. A $\mathcal{D}$-*tree* is a prefix closed subset $T \subseteq \mathcal{D}^*$ such that if $x \cdot c \in T$, where $x \in \mathcal{D}^*$ and $c \in \mathcal{D}$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For $x \in T$, the nodes $x \cdot c \in T$, where $c \in \mathcal{D}$, are the *successors* of x. A *leaf* is a node with no successors. A *path* of T is a set $\pi \subseteq T$ such that $\varepsilon \in T$ and, for every $x \in \pi$, either x is a leaf or there is a unique $c \in \mathcal{D}$ such that $x \cdot c \in \pi$. For an alphabet Σ, a Σ-labeled $\mathcal{D}$-tree is a pair $\langle T, V \rangle$ where $T \subseteq \mathcal{D}^*$ is a $\mathcal{D}$-tree and $V : T \to \Sigma$ maps each node of T to a symbol in Σ.

Alternating tree automata are a generalization of nondeterministic tree automata [26]. Intuitively, while a nondeterministic tree automaton that visits a node of the input tree sends exactly one copy of itself to each of the successors of

the node, an alternating automaton can send several copies of itself to the same successor. A *symmetric* alternating tree automaton [22,35] does not distinguish between the different successors of a node, and can send copies of itself only in a universal or an existential manner, possibly with ε-transitions. We use a partition of the state space of the automaton in order to denote the type of transitions from it. Formally, an *alternating parity tree automaton* (*APT*) is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, where Σ is a finite input alphabet; Q is a finite set of states, partitioned into universal ($Q^{\wedge}$), existential ($Q^{\vee}$), ε-and ($Q^{(\varepsilon,\wedge)}$), and ε-or ($Q^{(\varepsilon,\vee)}$) states (we also write $Q^{\vee,\wedge} = Q^{\vee} \cup Q^{\wedge}$, and $Q^{\varepsilon} = Q^{(\varepsilon,\vee)} \cup Q^{(\varepsilon,\wedge)}$); $q_0 \in Q$ is an initial state; $\delta : Q \times \Sigma \to (Q \cup 2^Q)$ is a transition function such that for all $\sigma \in \Sigma$, we have that $\delta(q,\sigma) \in Q$ for $q \in Q^{\vee,\wedge}$, and $\delta(q,\sigma) \in 2^Q$ for $q \in Q^{\varepsilon}$; and F is an acceptance condition, to be defined later. We assume that Q contains in addition two special states *ff* and *tt*, called *rejecting sink* and *accepting sink*, respectively, such that for all $a \in \Sigma$, we have that $\delta(tt,a) = tt$ and $\delta(ff,a) = ff$. The classification of *ff* and *tt* is arbitrary, and the acceptance condition F is defined in such a way that paths in a run tree of $\mathcal{A}$ that get stuck in the accepting (resp. rejecting) sink satisfy (resp. do not satisfy) F. Transitions from states in Q^{ε} launch copies of $\mathcal{A}$ that stay on the same input node as before the transition, while transitions from states in $Q^{\vee,\wedge}$ launch copies that advance to sons of the current node.

When a symmetric alternating tree automaton $\mathcal{A}$ runs on an input tree it starts with a copy in state q_0 whose reading head points to the root of the tree. It then follows δ in order to send further copies. For example, if a copy of $\mathcal{A}$ that is in state $q \in Q^{(\varepsilon,\vee)}$ is reading a node x labeled σ, and $\delta(q,\sigma) = \{q_1, q_2\}$, then this copy proceeds either to state q_1 or to state q_2, and its reading head stays in x. As another example, if $q \in Q^{\wedge}$ and $\delta(q,\sigma) = q_1$, then $\mathcal{A}$ sends a copy in state q_1 to every son of x. Note that different copies of $\mathcal{A}$ may have their reading head pointing to the same node of the input tree. Formally, a *run* of $\mathcal{A}$ on a Σ-labeled $\mathcal{D}$-tree $\langle T, V \rangle$ is a $(T \times Q)$-labeled $\mathbb{N}$-tree $\langle T_r, r \rangle$. A node in T_r labeled by (x,q) describes a copy of $\mathcal{A}$ in state q that reads the node x of T. A run has to satisfy $r(\varepsilon) = (\varepsilon, q_0)$ and, for all $y \in T_r$ with $r(y) = (x,q)$, the following hold:

- If $q \in Q^{\wedge}$ (resp. $q \in Q^{\vee}$) and $\delta(q, V(x)) = p$, then for each son (resp. for exactly one son) $x \cdot d$ of x, there is a node $y \cdot i \in T_r$ with $r(y \cdot i) = (x \cdot d, p)$.
- If $q \in Q^{(\varepsilon,\wedge)}$ (resp. $q \in Q^{(\varepsilon,\vee)}$) and $\delta(q, V(x)) = \{p_0, \ldots, p_k\}$, then for all $i \in \{0..k\}$ (resp. for one $i \in \{0..k\}$) the node $y \cdot i \in T_r$, and $r(y \cdot i) = (x, p_i)$.

A *parity condition* is a function $F : Q \to C$, where $C = \{C_{\min}, \ldots, C_{\max}\} \subset \mathbb{N}$ is a set of colors. We assume that $F(tt)$ is even, and that $F(ff)$ is odd. Consider a run $\langle T_r, r \rangle$. A path $\pi \subseteq T_r$ satisfies the acceptance condition F iff the maximal color appearing infinitely often in the coloring of the states labeling π is even. Formally, let $inf(r|\pi) \subseteq Q$ be the set of states that r visits infinitely often along π. Thus, $q \in inf(r|\pi)$ iff there are infinitely many $y \in \pi$ such that $r(y) \in T \times \{q\}$. Then, $maxC(\pi) = \max_{q \in inf(r|\pi)} F(q)$, and π satisfies F if $maxC(\pi)$ is even. The size $|C|$ of C is called the *index* of the automaton. A run $\langle T_r, r \rangle$ is accepting if all its paths satisfy F. The automaton $\mathcal{A}$ accepts an input tree $\langle T, V \rangle$ if there is an accepting run of $\mathcal{A}$ on $\langle T, V \rangle$. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of Σ-labeled D-trees accepted by $\mathcal{A}$. We say that an automaton $\mathcal{A}$ is nonempty iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$. Note that since $\mathcal{A}$ is symmetric, the set D of directions of the trees plays no role in the definition of a run.

2.3. Automata for temporal logics

A wide range of branching-time temporal logics can be translated to alternating tree automata. Among the others, here we consider CTL, CTL*, the modal μ-calculus, and the alternation-free μ-calculus (for a full definition of the syntax and the semantics of these logics, see [23]). The size of the automaton, as well as its acceptance condition depend on the particular logic.

Theorem 1. *(See [15,23].) Given a temporal-logic formula φ, it is possible to construct a tree automaton (either symmetric or asymmetric) $\mathcal{A}_\varphi$ such that $\mathcal{L}(\mathcal{A}_\varphi)$ is exactly the set of trees satisfying φ. Moreover*:

- *If φ is a* CTL *or an alternation-free μ-calculus formula, then $\mathcal{A}_\varphi$ is an alternating parity automaton with $O(|\varphi|)$ states and index* 2.
- *If φ is a* CTL* *formula, then $\mathcal{A}_\varphi$ is an alternating parity automaton with $2^{O(|\varphi|)}$ states and index* 3.
- *If φ is a μ-calculus formula, then $\mathcal{A}_\varphi$ is an alternating parity automaton with $O(|\varphi|)$ states and index $O(|\varphi|)$.*

It is worth noting that for all the automata $\mathcal{A}_\varphi$ mentioned above have the special property that the only transitions that actually depend on the input letter are transitions that go to the accepting or rejecting sinks, i.e., for every $a, a' \in \Sigma$, and every $q \in Q$, we have that $\delta(q,a) \neq \delta(q,a')$ iff $\delta(q,a), \delta(q,a') \in \{tt, ff\}$.

3. The hierarchical model-checking game

The game-based approach to model checking a flat system $\mathcal{K}$, with respect to a branching-time temporal logic specification φ, reduces the model-checking problem to solving a game obtained by taking the product of $\mathcal{K}$ with the alternating tree automaton $\mathcal{A}_\varphi$ [23]. In this section, we extend this approach to hierarchical structures: given a hierarchical system $\mathcal{K}$ and an alternating tree automaton $\mathcal{A}$, we construct a game $\mathcal{G}_{\mathcal{K},\mathcal{A}}$, such that Player 0 wins the game iff the tree obtained

by unwinding the flat expansion of $\mathcal{K}$ is accepted by $\mathcal{A}$. In particular, when $\mathcal{A}$ accepts exactly all the tree models of a branching-time formula φ, the above holds iff $\mathcal{K}$ satisfies φ. Note that a naive approach for doing this is to start by constructing the flat expansion of $\mathcal{K}$ and then applying [23]. The whole point, however, is to avoid the exponentially large flat system and work directly in the hierarchical setting. We focus on the case in which $\mathcal{A}$ is an alternating parity tree automaton (APT), to which μ-calculus formulas are translated.

Given a hierarchical system $\mathcal{K} = \langle \mathcal{K}_1, \ldots, \mathcal{K}_n \rangle$ and an APT $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, the hierarchical two-player game $\mathcal{G}_{\mathcal{K},\mathcal{A}} = (\mathcal{V}, \Gamma)$ for $\mathcal{K}$ and $\mathcal{A}$ is defined as follows. The hierarchical arena $\mathcal{V}$ has a sub-arena $\mathcal{V}_{i,q}$ for every $2 \leqslant i \leqslant n$ and state $q \in Q$, which is essentially the product of the structure $\mathcal{K}_i$ with $\mathcal{A}$, where the initial state of $\mathcal{K}_i$ is paired with the state q of $\mathcal{A}$. For $i = 1$, we need only the sub-arena V_{1,q_0}. The hierarchical order of the sub-arenas is consistent with the one in $\mathcal{K}$. Thus, the sub-arena $\mathcal{V}_{i,q}$ can be referred to by boxes of sub-arena $\mathcal{V}_{j,p}$ only if $i > j$. Let $\mathcal{K}_i = \langle AP, W_i', \mathcal{B}_i', in_i', exit_i', \tau_i', \mathcal{R}_i', \sigma_i' \rangle$ and let $\mathcal{A} = \langle 2^{AP}, Q, q_0, \delta, F \rangle$ be an APT with Q partitioned to $Q^{(\varepsilon,\wedge)}$, $Q^{(\varepsilon,\vee)}$, $Q^{\wedge}$, and $Q^{\vee}$. Then, the sub-arena $\mathcal{V}_{i,q} = \langle W_{i,q}^0, W_{i,q}^1, \mathcal{B}_{i,q} in_{i,q}, exit_{i,q}, \tau_{i,q}, \mathcal{R}_{i,q} \rangle$ is defined as follows.

- $W_{i,q}^0 = W_i' \times (Q^{\vee} \cup Q^{(\varepsilon,\vee)})$, $W_{i,q}^1 = W_i' \times (Q^{\wedge} \cup Q^{(\varepsilon,\wedge)})$, $in_{i,q} = (in_i', q)$, and $exit_{i,q} = exit_i' \times Q^{\vee,\wedge}$.
- $\mathcal{B}_{i,q} = \mathcal{B}_i' \times Q$, and $\tau_{i,q}(b, q) = (\tau_i'(b), q)$.
- For a state $u = (w, \hat{q}) \in W_i' \times Q$, if $\hat{q} \in Q^{\varepsilon}$ and $\delta(\hat{q}, \sigma_i'(w)) = \{p_0, \ldots, p_k\}$, then $(u, v) \in \mathcal{R}_{i,q}$ iff $v \in \{(w, p_0), \ldots, (w, p_k)\}$; and if $\hat{q} \in Q^{\vee,\wedge}$, then $(u, v) \in \mathcal{R}_{i,q}$ iff $v = (w', \delta(\hat{q}, \sigma_i'(w)))$ and $(w, w') \in \mathcal{R}_i'$.
- For $(b, p) \in \mathcal{B}_i' \times Q$, and an exit $(e, \hat{q}) \in exit'_{\tau_i'(b)} \times Q^{\vee,\wedge}$ of this box, then $(((b, p), (e, \hat{q})), v) \in \mathcal{R}_{i,q}$ iff $v = (w', \delta(\hat{q}, \sigma'_{\tau_i'(b)}(e)))$ and $((b, e), w') \in \mathcal{R}_i'$.

The winning condition of the game $\mathcal{G}_{\mathcal{K},\mathcal{A}}$ is induced by the acceptance condition of $\mathcal{A}$. Formally, for each state (w, q) of $\mathcal{V}_{i,q}$, we have $\Gamma(w, q) = F(q)$.

We now argue that the model-checking problem $\mathcal{K} \models \varphi$ can be reduced to solving the hierarchical game $\mathcal{G}_{\mathcal{K},\mathcal{A}_\varphi}$. For that, we show that $\mathcal{G}_{\mathcal{K},\mathcal{A}_\varphi}$ is equivalent to the flat game $\mathcal{G}_{\mathcal{K}^{\mathrm{f}},\mathcal{A}_\varphi}$. Since, by [23], the model-checking problem can be reduced to solving the latter, we are done. The proof of the equivalence between $\mathcal{G}_{\mathcal{K},\mathcal{A}_\varphi}$ and $\mathcal{G}_{\mathcal{K}^{\mathrm{f}},\mathcal{A}_\varphi}$ is based on a bijection between strategies of one game and strategies of the other. In particular, for every winning strategy for one of the players in $\mathcal{G}_{\mathcal{K},\mathcal{A}}$, there is a corresponding winning strategy for the same player in $\mathcal{G}_{\mathcal{K}^{\mathrm{f}},\mathcal{A}}$, and vice versa.

Given a hierarchical system $\mathcal{K} = \langle \mathcal{K}_1, \ldots, \mathcal{K}_n \rangle$, and an APT $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, consider a state $u = (b_1, \ldots, b_h, w)$ of $\mathcal{K}^{\mathrm{f}}$. Observe that for every state q of $\mathcal{A}$, there is a node $((b_1, \ldots, b_h, w), q)$ in the arena of $\mathcal{G}_{\mathcal{K}^{\mathrm{f}},\mathcal{A}}$ which represents a copy of $\mathcal{A}$ that is at state q and is reading u. On the other hand, the flat expansion of the arena of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$ is richer, and it has a node $((b_1, q_1), \ldots, (b_h, q_h), (w, q))$ for every sequence $q_1, \ldots, q_h, q$ of states of $\mathcal{A}$. As before, such a node represents a copy of $\mathcal{A}$ that is at state q and is reading u. However, it also remembers for each of the boxes $b_1, \ldots, b_h$, the states $q_1, \ldots, q_h$ that this copy of the automaton was at when it last entered each of these boxes. It is easy to see that every initial play of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$ can be transformed into a play of $\mathcal{G}_{\mathcal{K}^{\mathrm{f}},\mathcal{A}}$ by simply dropping this extra information from every node. Note that the reverse is also possible since given a node on an initial play of $\mathcal{G}_{\mathcal{K}^{\mathrm{f}},\mathcal{A}}$, the states at which the copy of the automaton was, when it entered the various boxes encoded in this node, can be recovered from previous nodes of the play. The following lemma formally describes this association between initial plays of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$ and initial plays of $\mathcal{G}_{\mathcal{K}^{\mathrm{f}},\mathcal{A}}$.

Lemma 1. *There is a bijection* expand *from initial plays of $\mathcal{G}_{\mathcal{K}^{\mathrm{f}},\mathcal{A}}$ to initial plays of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$, such that* expand *preserves the winner of maximal plays. Moreover, $\hat{\pi}$ is an extension of π iff* expand$(\hat{\pi})$ *is an extension of* expand(π).

Proof. Consider an initial play $\pi = \pi_0, \pi_1, \ldots$ of $\mathcal{G}_{\mathcal{K}^{\mathrm{f}},\mathcal{A}}$, and note that for every i we have that $\pi_i = \langle (b_{i,1}, \ldots, b_{i,h_i}, w_i), q_i \rangle$, where $b_{i,1}, \ldots, b_{i,h_i}$ are boxes of $\mathcal{K}$, w_i is a state of $\mathcal{K}$, and q_i is a state of $\mathcal{A}$. Let $Qin(i, j)$ be the state q_m, where m is the largest index such that $b_{i,j} = b_{m,h_m}$ but $b_{i,j} \neq b_{m-1,h_{m-1}}$. That is, m is the last time that π entered the box $b_{i,j}$. Since π starts at the entry of the top-level arena, m is well defined. Let $\mathsf{expand}(\pi, \pi_i) = \langle (b_{i,1}, Qin(i, 1)), \ldots, (b_{i,h_i}, Qin(i, h_i)), (w_i, q_i) \rangle$, and let $\mathsf{expand}(\pi) = \mathsf{expand}(\pi, \pi_0), \mathsf{expand}(\pi, \pi_1), \ldots$. Observe that $\mathsf{expand}(\pi)$ is an initial play of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$, and that if $\hat{\pi}$ is an extension of π, then $\mathsf{expand}(\hat{\pi})$ is an extension of $\mathsf{expand}(\pi)$. For the reverse mapping, let $s = \langle (b_1, q_1), \ldots, (b_h, q_h), (w, q) \rangle$ be a node of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$, and let $\mathsf{contract}(s) = \langle (b_1, \ldots, b_h, w), q \rangle$ be the projection of this node on the arena of $\mathcal{G}_{\mathcal{K}^{\mathrm{f}},\mathcal{A}}$.

It is easy to see that the mapping that maps every path π to $\mathsf{expand}(\pi)$ is one to one. Hence, to show that expand is a bijection, and contract is its inverse, it is enough to show that for every initial play π' of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$ we have that $\mathsf{expand}(\mathsf{contract}(\pi')) = \pi'$. Consider then an initial play π' of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$, and assume by way of contradiction that $\mathsf{expand}(\mathsf{contract}(\pi')) \neq \pi'$. Let $\pi = \pi_0, \pi_1, \ldots$ be the contraction of π', and for every $i \geqslant 0$ let $\pi_i = \langle (b_{i,1}, \ldots, b_{i,h_i}, w_i), q_i \rangle$. Let j be the largest index for which $\mathsf{expand}(\pi, \pi_j) = \pi_j'$, and observe that since $\mathsf{expand}(\pi, \pi_0) = \langle in_1, q_0 \rangle = \pi_0'$, then $j > 0$. Let $\pi_j' = \mathsf{expand}(\pi, \pi_j) = \langle (b_{j,1}, Qin(j, 1)), \ldots, (b_{j,h_j}, Qin(j, h_j)), (w_j, q_j) \rangle$. By the definition of the edges of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$, there are four options for the move from π_j' to π_{j+1}':

- The token remained in the same sub-arena as w_j. In this case, $h_{j+1} = h_j$, and $\pi_{j+1}' = \langle (b_{j,1}, Qin(j, 1)), \ldots, (b_{j,h_j}, Qin(j, h_j)), (w_{j+1}, q_{j+1}) \rangle$. It follows that for every $1 \leqslant l \leqslant h_{j+1}$ we have that $b_{j,l} = b_{j+1,l}$ and $Qin(j, l) = Qin(j + 1, l)$.

- The token exited the sub-arena of w_j and returned to the calling sub-arena: In this case, $h_{j+1} = h_j - 1$, and $\pi'_{j+1} = \langle (b_{j,1}, Qin(j,1)), \ldots, (b_{j,h_j-1}, Qin(j,h_j-1)), (w_{j+1}, q_{j+1})\rangle$. It follows that for every $1 \leqslant l \leqslant h_{j+1}$ we have that $b_{j,l} = b_{j+1,l}$ and $Qin(j,l) = Qin(j+1,l)$.
- The token entered a new box (b,q). In this case, $h_{j+1} = h_j + 1$, and $\pi'_{j+1} = \langle (b_{j,1}, Qin(j,1)), \ldots, (b_{j,h_j}, Qin(j,h_j)), (b,q), (w_{j+1}, q)\rangle$. Note that since $\mathcal{K}$ is hierarchical (and not recursive), b cannot be equal to any of the boxes $b_{j,1}, \ldots, b_{j,h_j}$. It follows that for every $1 \leqslant l \leqslant h_j$ we have that $b_{j,l} = b_{j+1,l}$ and $Qin(j,l) = Qin(j+1,l)$.
- The token exited the sub-arena of w_j and immediately entered a new box (b,q). In this case, $h_{j+1} = h_j$, and $\pi'_{j+1} = \langle (b_{j,1}, Qin(j,1)), \ldots, (b_{j,h_j-1}, Qin(j,h_j-1)), (b,q), (w_{j+1}, q)\rangle$. Note that since $\mathcal{K}$ is hierarchical (and not recursive), b cannot be equal to any of the boxes $b_{j,1}, \ldots, b_{j,h_j-1}$. It follows that for every $1 \leqslant l \leqslant h_j - 1$ we have that $b_{j,l} = b_{j+1,l}$ and $Qin(j,l) = Qin(j+1,l)$.

It is not hard to see that in all cases it must be that $\mathsf{expand}(\pi, \pi_{j+1}) = \pi'_{j+1}$, which is a contradiction to our contrapositive assumption that $\mathsf{expand}(\pi, \pi_{j+1}) \neq \pi'_{j+1}$. It follows that expand is a bijection and that contract is its inverse. It is left to show that the two mappings preserve the winner of maximal plays. Given a maximal initial play π of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$, note that the winning condition of $\mathcal{G}_{\mathcal{K},\mathcal{A}}$ refers only to the sequence of automaton states $q_0, q_1, \ldots$, taken from the right component of the last pair in the nodes of π, and the winning condition of $\mathcal{G}_{\mathcal{K}^f,\mathcal{A}}$ refers to the same sequence of automaton states as found in the nodes of $\mathsf{contract}(\pi)$. Hence, contract preserves the winner of maximal plays, and, being its inverse, so does expand. □

Theorem 2. *Consider a hierarchical system $\mathcal{K}$ and a branching-time formula φ. The following are equivalent*: (*i*) *$\mathcal{K}$ satisfies φ.* (*ii*) *Player* 0 *has a winning strategy in the flat game $\mathcal{G}_{\mathcal{K}^f,\mathcal{A}_\varphi}$.* (*iii*) *Player* 0 *has a winning strategy in the hierarchical game $\mathcal{G}_{\mathcal{K},\mathcal{A}_\varphi}$.*

Proof. As stated before, the equivalence of (*i*) and (*ii*) is proven in [23]. To prove the equivalence of (*ii*) and (*iii*), we show that for every winning strategy for one of the players in $\mathcal{G}_{\mathcal{K},\mathcal{A}_\varphi}$ there is a corresponding winning strategy of the same player in $\mathcal{G}_{\mathcal{K}^f,\mathcal{A}_\varphi}$, and vice versa.

Given $\sigma \in \{0,1\}$ and a strategy ξ' for Player σ in $\mathcal{G}_{\mathcal{K},\mathcal{A}_\varphi}$, Lemma 1 implies that the strategy ξ defined by $\xi(\pi) = s$, where s is the last node in the play $\mathsf{contract}(\mathsf{expand}(\pi) \cdot \xi'(\mathsf{expand}(\pi)))$, is a strategy for Player σ in $\mathcal{G}_{\mathcal{K}^f,\mathcal{A}_\varphi}$. Observe that if π' is a maximal play according to ξ', then $\mathsf{contract}(\pi')$ is a maximal play according to ξ. Hence, since contract is a bijection that preserves the winner of maximal plays, it follows that ξ' is winning for Player σ in $\mathcal{G}_{\mathcal{K},\mathcal{A}_\varphi}$ iff ξ is winning for Player σ in $\mathcal{G}_{\mathcal{K}^f,\mathcal{A}_\varphi}$. A symmetric argument shows that ξ is a winning strategy for Player σ in $\mathcal{G}_{\mathcal{K}^f,\mathcal{A}_\varphi}$, iff the strategy ξ' defined by $\xi'(\pi') = s'$, where s' is the last node in the play $\mathsf{expand}(\mathsf{contract}(\pi') \cdot \xi(\mathsf{contract}(\pi')))$, is a winning strategy for Player σ in $\mathcal{G}_{\mathcal{K},\mathcal{A}_\varphi}$. □

In Section 4, we solve hierarchical two-player games and show how Theorem 2 leads to optimal model-checking algorithms for hierarchical systems.

4. Solving hierarchical parity games

In this section we present an algorithm for solving hierarchical parity games. Consider a game $\mathcal{G} = (\mathcal{V}, \Gamma)$. A naive algorithm for solving the game would generate the flat expansion of $\mathcal{V}$ and solve it. In the flat expansion, each sub-arena may appear in many different contexts. The idea behind our algorithm is that even though the sub-arena appears in different contexts, the effect of the strategies chosen by the players for the segment of the game inside the sub-arena is independent of the context and can be summarized efficiently. The effects of every strategy of Player 0 for the segment of the play inside a sub-arena $\mathcal{V}_i$, can be captured by a *summary function* mapping each exit of $\mathcal{V}_i$ to the best color that Player 1 can hope for, if he chooses to respond by directing the token to leave $\mathcal{V}_i$ through this exit. The algorithm for solving the game $\mathcal{G} = (\mathcal{V}, \Gamma)$ then solves a sequence of flat parity games, obtained by replacing sub-arenas by gadgets that represent the behavior of Player 0 as a choice among the possible summary functions, and the behavior of Player 1 as a choice of the exit through which he wants the token to exit the sub-arena. The gadgets also take into account the possibility that the game will stay forever in the sub-arena.

We now describe the concept of summary functions in detail. Consider first a play that enters a box that has a single exit. Each player has one goal that is independent of the context in which the box appears: to either win inside the box, or failing that, use a strategy that provides the biggest possible advantage over the segment of the play that goes through the box. In the case where the box has multiple exits, the situation is more involved: if a player cannot force a win inside the box, he is faced with the question of which exit he should try to force the play to exit through. Depending on the context in which the box appears, it may be beneficial to force the play to a specific exit even if that involves letting the other player gain the upper hand in the path leading to it. Also, in certain situations, none of the players may force the game to a specific exit, and the strategy a player chooses may reflect a certain tradeoff between the different colors achieved on the paths going to the different exits.

In order to describe the relative merit of colors, we define an ordering $\succcurlyeq_0$ on colors by letting $c \succcurlyeq_0 c'$ when c is better for Player 0 than c'. Formally, $c \succcurlyeq_0 c'$ if the following holds: if c' is even then c is even and $c \geqslant c'$; and if c' is odd then

Algorithm 1: Solving a hierarchical parity game.

Input: $\mathcal{G} = (V, \Gamma)$, where $\mathcal{V} = \langle \mathcal{V}_1, \ldots, \mathcal{V}_n \rangle$
Output: true iff Player 0 wins $\mathcal{G}$
for $i = n$ **downto** 1 **do**
  $M_i = \emptyset$
  forall $g \in Summ(\mathcal{V}_i)$ **do**
    $\mathcal{G}^s_{i,g} = \mathsf{loop}(g, \mathsf{simplify}(\mathcal{V}_i, H_{i+1}, \ldots, H_n))$
    if *Player* 0 *wins* $\mathcal{G}^s_{i,g}$ **then** $M_i = M_i \cup \{g\}$
  end
  if $i > 1$ **then** construct H_i from $\mathcal{V}_i$ and M_i
end
return true iff $M_1 \neq \emptyset$

either c is even, or c is also odd and $c \leqslant c'$. We denote by $\min^{\succcurlyeq_0}$ ($\max^{\succcurlyeq_0}$) the operation of taking the minimal (maximal) color, according to $\succcurlyeq_0$, of a finite set of colors. Consider a strategy ξ of Player 0 for a sub-arena $\mathcal{V}_i$. We define a function $g_\xi : exit_i \to C \cup \{\dashv\}$, called the *summary function* of ξ, that summarizes the best responses of Player 1 to ξ.[6] Let $e \in exit_i$ be an exit node of $\mathcal{V}_i$. If ξ is such that no matter how Player 1 plays, the token never exits through e, then we set $g_\xi(e) = \dashv$. Otherwise, we set $g_\xi(e)$ to be the most beneficial color that Player 1 can achieve along all plays that agree with ξ and exit through e. Formally, let $plays(\xi, e)$ be the set of all plays in $\mathcal{V}_i$ that agree with ξ and exit through e. For every $e \in exit_i$ we define $g_\xi(e) = \dashv$ if $plays(\xi, e) = \emptyset$, and $g_\xi(e) = \min^{\succcurlyeq_0}\{maxC(\pi) \colon \pi \in plays(\xi, e)\}$.

Recall that if ξ is not a losing strategy for Player 0 then all plays that agree with ξ and remain inside $\mathcal{V}_i$ are winning for Player 0. Hence, if ξ is not a losing strategy then Player 1 will always direct the token to exit through some exit $e \in exit_i$. Note that Player 1 can only choose e for which $g_\xi(e) \neq \dashv$, and that the choice of e depends on the context in which the sub-arena $\mathcal{V}_i$ appears. A key point in our algorithm is that, for every game $\mathcal{G}$ in which the sub-arena $\mathcal{V}_i$ is used, and every Player 0 strategy ξ for $\mathcal{V}_i$, if ξ is not a losing strategy then g_ξ captures all the information needed to analyze the influence of the play inside $\mathcal{V}_i$ on $\mathcal{G}$.

Let $Summ(\mathcal{V}_i) = \{g \colon g \text{ is a function from } exit_i \text{ to } C \cup \{\dashv\}\}$ be the set of all summary functions[7] for strategies of Player 0 over $\mathcal{V}_i$. If $\mathcal{V}_i$ has no exits, then $Summ(\mathcal{V}_i)$ contains only the empty summary function ε. Based on the ordering $\succcurlyeq_0$ we defined for colors, we can define a partial order $\succcurlyeq$ on $Summ(\mathcal{V}_i)$, by letting $g \succcurlyeq g'$ if for every exit node e of $\mathcal{V}_i$ the following holds: $g(e) = \dashv$, or $g(e) \neq \dashv \neq g'(e)$ and $g(e) \succcurlyeq_0 g'(e)$. Observe that if ξ and ϱ are two Player 0 strategies that are not losing strategies, and $g_\xi \succcurlyeq g_\varrho$, then Player 0 can always choose ξ over ϱ. Given a summary function $g \in Summ(\mathcal{V}_i)$, we say that a strategy ξ of Player 0 *achieves* g if $g_\xi \succcurlyeq g$; we say that g is *feasible* if there is a strategy ξ that achieves it; and we say that g is *relevant* if it can be achieved by a memoryless strategy that is not losing. In particular, if $\mathcal{V}_i$ has no exits, deciding whether the empty summary function ε is relevant amounts to deciding if it is not losing, i.e., to solving the game over $\mathcal{V}_i$.

We now describe the algorithm for solving a hierarchical parity game. The outline of the algorithm is described in Algorithm 1. Given a hierarchical parity game $\mathcal{G} = (\mathcal{V}, \Gamma)$, where $\mathcal{V} = \langle \mathcal{V}_1, \ldots, \mathcal{V}_n \rangle$, our algorithm solves $\mathcal{G}$ by working its way up the hierarchy, starting with the lowest level sub-arena $\mathcal{V}_n$. At iteration $n \geqslant i \geqslant 1$, the algorithm first calculates the set M_i of relevant summary functions for strategies of Player 0 over $\mathcal{V}_i$. It does so by going over all summary functions and checking their relevancy. In order to check whether a summary function g is relevant, the algorithm solves a simple parity game $\mathcal{G}^s_{i,g} = (\mathcal{V}^s_{i,g}, \Gamma^s_{i,g})$, which is defined in such a way that g is relevant iff Player 0 has a winning strategy for $\mathcal{G}^s_{i,g}$. The arena $\mathcal{V}^s_{i,g}$ is built from $\mathcal{V}_i$ by applying to it two operations: simplify, and loop. Once the set M_i is found, the algorithm uses it in order to construct a 3-level DAG structure H_i that reflects Player 0's choice of strategy for the sub-arena $\mathcal{V}_i$, and Player 1's possible responses to this strategy. The gadget H_i, together with $H_{i+1}, \ldots, H_n$ which were constructed in previous iterations, is used in future iterations. Indeed, as detailed below, the essence of the simplify procedure is to replace a box that refers to a sub-arena $\mathcal{V}_j$ by the gadget H_j. Since the top-level arena $\mathcal{V}_1$ has no exits, the only summary function it has is the empty summary function ε, which, by definition, is relevant iff Player 0 wins $\mathcal{G}$. Hence, the algorithm reduces the problem of solving the hierarchical game $\mathcal{G}$ to the problem of solving the simple parity game $\mathcal{G}^s_{1,\varepsilon}$.

We now describe the construction of the gadget H_i. Let M_i be the set of all relevant summary functions for $\mathcal{V}_i$. Then, H_i is the following 3-level DAG:

- The set of nodes of H_i is $\{p\} \cup M_i \cup (exit_i \times C)$. The node p is a Player 0 node, every $g \in M_i$ is a Player 1 node, and a node $(e, c) \in exit_i \times C$ belongs to the same player that e belongs to.
- The set of edges is $\bigcup_{g \in M_i}(\{(p, g)\} \cup \{(g, (e, g(e))) \colon e \in exit_i \wedge g(e) \neq \dashv\})$.
- A node $(e, c) \in exit_i \times C$ is colored by c. These are the only colored nodes.

Finally, we remove from H_i all the nodes that are not reachable from its root p. Thus, in particular, if $M_i = \emptyset$, then p is the only node that remains in H_i. Intuitively, when the token is at the root p of the gadget H_i, Player 0 chooses

[6] Note that our choice to consider summary functions of Player 0 strategies is arbitrary, and we could have taken Player 1's point of view instead.
[7] We call every $g \in Summ(\mathcal{V}_i)$ a "summary function" even if there is no Player 0 strategy whose summary is g.

a relevant summary function g for $\mathcal{V}_i$, and moves the token to the node g. In response, Player 1 chooses an exit $e \in exit_i$ for which $g(e) \neq \dashv$, and moves the token to the node $(e, g(e))$. The color of $(e, g(e))$ is $g(e)$, which is the best possible color achievable by Player 1 in any play over $\mathcal{V}_i$ that exits through e, when playing against a Player 0 strategy that achieves g.

Observe that if $M_i = \emptyset$, then it must be that all the summary functions in $Summ(\mathcal{V}_i)$ are not relevant, i.e., that all Player 0 strategies for $\mathcal{V}_i$ are losing. Note that this behavior is preserved if we turn all exit nodes of $\mathcal{V}_i$ to non-exit nodes. Hence, from the determinacy of simple parity games it follows that Player 1 has a winning strategy for $\mathcal{V}_i$, which explains why in this case H_i is a single terminal Player 0 node. Recall that for every $g \in M_i$ there exists at least one non-losing Player 0 strategy ξ^g that achieves g, and that since ξ^g is not losing, every play that agrees with ξ^g and does not exit $\mathcal{V}_i$ is winning for Player 0. It follows that if for every $e \in exit_i$ we have $g(e) = \dashv$ (in particular, if $exit_i = \emptyset$), then every play that is consistent with ξ^g cannot exit $\mathcal{V}_i$, and is thus winning for Player 0. This explains why in such a case the node g is a terminal Player 1 node.

It is left to describe and explain the operations simplify and loop. We start with simplify, which *simplifies* a hierarchical arena $\mathcal{V}_i$ by replacing every box $b \in \mathcal{B}_i$ by a copy of the gadget $H_{\tau_i(b)}$. Observe that the hierarchical nesting of the sub-arenas guarantees that all the boxes in $\mathcal{B}_i$ refer to arenas with an index higher than i, and thus the gadgets required for replacing them were already constructed in previous iterations. We usually denote the resulting flat arena $\mathsf{simplify}(\mathcal{V}_i, H_{i+1}, \ldots, H_n)$ by the shorter notation $\mathcal{V}_i^s$. We now formally define $\mathcal{V}_i^s$. To prevent name clashes between copies of the same gadget, given a box $b \in \mathcal{B}_i$, let H^b be a copy of $H_{\tau_i(b)}$ with all nodes renamed by annotating them with b. Replacing b with the gadget H^b is done by replacing every transition $(u, b) \in \mathcal{R}_i$ that enters b with a transition (u, p^b) that goes to the root of H^b, and replacing every transition $((b, e), v) \in \mathcal{R}_i$ that exits b with one transition $((e, c)^b, v)$ for every color c for which $(e, c)^b$ is present in H^b. Formally, given $\mathcal{V}_i = \langle W_i^0, W_i^1, \mathcal{B}_i, in_i, exit_i, \tau_i, \mathcal{R}_i \rangle$, then $\mathcal{V}_i^s = \langle W_i^{0^s}, W_i^{1^s}, \emptyset, in_i, exit_i, \emptyset, \mathcal{R}_i^s \rangle$, and its coloring function $\Gamma_i^s : W_i^s \to C$ are as follows:

- For $\sigma \in \{0, 1\}$, we have that $W_i^{\sigma\, s} = W_i^\sigma \cup \bigcup_{b \in \mathcal{B}_i} H^{b,\sigma}$, where $H^{b,\sigma}$ is the set of Player σ nodes of H^b.
- $\mathcal{R}_i^s$ is $(W_i^s \times W_i^s) \cap \langle \bigcup_{b \in \mathcal{B}_i} (\{(u, p^b) \colon (u, b) \in \mathcal{R}_i\} \cup \{((e, c)^b, v) \colon c \in C,\ e \in exit_{\tau_i(b)},\ ((b, e), v) \in \mathcal{R}_i\} \cup R(H^b)) \cup \mathcal{R}_i \rangle$, with $R(H^b)$ being the set of transitions of H^b.
- $\Gamma_i^s(s) = \Gamma(s)$ for $s \in W_i$ for which $\Gamma(s)$ is defined; for every $b \in \mathcal{B}_i$ and every $(e, c) \in exit_{\tau_i(b)} \times C$ we have $\Gamma_i^s((e, c)^b) = c$; otherwise, $\Gamma_i^s(s)$ is undefined.

We now describe the operation loop, used in the process of identifying relevant summary functions. Given a summary function g over a sub-arena $\mathcal{V}_i$, the operation $\mathsf{loop}(g, \mathcal{V}_i^s)$ constructs a simple arena $\mathcal{V}_{i,g}^s$ such that Player 0 wins the associated simple parity game $\mathcal{G}_{i,g}^s = (\mathcal{V}_{i,g}^s, \Gamma_{i,g}^s)$ iff g is relevant. Since the modifications done by loop to its input arena do not concern any of its boxes (if present), the operations simplify and loop, commute. I.e., $\mathsf{loop}(g, \mathsf{simplify}(\mathcal{V}_i, H_{i+1}, \ldots, H_n)) = \mathsf{simplify}(\mathsf{loop}(g, \mathcal{V}_i), H_{i+1}, \ldots, H_n)$.

Let $\mathcal{V}_{i,g} = \mathsf{loop}(g, \mathcal{V}_i)$. To construct $\mathcal{V}_{i,g}$ from $\mathcal{V}_i$, we add for every exit node $e \in exit_i$ a new Player 0 node $(e, 0)$. We color it by $g(e) + 1$ if $g(e)$ is odd, we color it by $g(e) - 1$ if $g(e)$ is even, and we leave it uncolored if $g(e) = \dashv$. Also, if $g(e) \neq \dashv$, we add an edge $(e, (e, 0))$, and an edge $((e, 0), in_i)$. Finally, we set all states of $\mathcal{V}_{i,g}$ as non-exits. Formally, given $\mathcal{V}_i = \langle W_i^0, W_i^1, \mathcal{B}_i, in_i, exit_i, \tau_i, \mathcal{R}_i \rangle$, then $\mathcal{V}_{i,g} = \langle W_{i,g}^0, W^1, \mathcal{B}_i, in_i, \emptyset, \tau_i, \mathcal{R}_{i,g} \rangle$ and its associated coloring function $\Gamma_{i,g} : W_{i,g} \to \{C_{\min} - 1, \ldots, C_{\max} + 1\}$ are as follows:

- $W_{i,g}^0 = W_i^0 \cup (exit_i \times \{0\})$.
- $\mathcal{R}_{i,g} = \mathcal{R}_i \bigcup_{e \in exit_i \wedge g(e) \neq \dashv} (\{(e, (e, 0))\} \cup \{((e, 0), in_i)\})$.
- $\Gamma_{i,g}(s) = \Gamma(s)$ for $s \in W_i^0 \cup W_i^1$ for which $\Gamma(s)$ is defined; $\Gamma_{i,g}(e, 0) = g(e) + 1$ if $g(e)$ is odd, and $\Gamma_{i,g}(e, 0) = g(e) - 1$ if $g(e)$ is even; otherwise, $\Gamma_{i,g}(s)$ is undefined.

Note that if $\mathcal{V}_i$ has no exits then it has only the empty summary function ε, and that $\mathcal{V}_i^s = \mathcal{V}_{i,\varepsilon}^s$. Thus, in particular, $\mathcal{V}^s = \mathcal{V}_{1,\varepsilon}^s$.

Lemma 2. *Given a summary function $g \in Summ(\mathcal{V}_i)$, over a sub-arena $\mathcal{V}_i$, we have that g is relevant iff Player 0 wins the game $\mathcal{G}_{i,g}$.*

Proof. For a sub-arena $\mathcal{V}_i$, and a summary function $g \in Summ(\mathcal{V}_i)$, let $\mathcal{V}_{i,g} = \mathsf{loop}(g, \mathcal{V}_i)$ as defined above. Observe that by replacing every move that exits $\mathcal{V}_i$ through some exit $e \in exit_i$, by a move to the node $(e, 0)$, every memoryless Player 0 strategy ξ over $\mathcal{V}_i$ induces a memoryless Player 0 strategy ξ' over $\mathcal{V}_{i,g}$, and vice versa. We prove the lemma by showing that ξ is not losing and achieves g iff ξ' is winning.

Assume first that ξ' is winning. Thus, in particular, all plays consistent with ξ' that do not visit a node of the form $(e, 0)$ are winning for Player 0. It follows that all plays consistent with ξ that do not exit $\mathcal{V}_i$ are winning for Player 0, which implies that ξ is not losing. It remains to show that the summary function g_ξ of ξ is such that $g_\xi \succcurlyeq g$. Consider first an exit $e \in exit_i$ such that $g(e) = \dashv$. Since in this case $(e, 0)$ is a terminal Player 0 node, it follows that no play consistent with ξ' can reach $(e, 0)$. Hence, no play consistent with ξ can exit $\mathcal{V}_i$ through e, and $g_\xi(e) = \dashv$. Consider now the case where $g(e) \neq \dashv$. Note that in order to show that $g_\xi \succcurlyeq g$, we have to show that $g_\xi(e) \succcurlyeq^0 g(e)$, i.e., that for every play $\pi = in_i \cdot \pi_1 \cdots \pi_k \cdot e \cdot out$

consistent with ξ that exits through e we have that $maxC(\pi) \succcurlyeq^0 g(e)$. Let π be such a play, and let c' be the color of $(e,0)$. Observe that the play $\pi' = (in_i \cdot \pi_1 \cdots \pi_k \cdot e \cdot (e,0))^\omega$ is consistent with ξ', and since ξ' is winning then $maxC(\pi')$ is even.

Observe that by the structure of π and π', either $maxC(\pi) = maxC(\pi') > c'$, or $maxC(\pi) \leqslant maxC(\pi') = c'$. Consider first the case where $maxC(\pi) = maxC(\pi') > c'$. If $g(e)$ is odd, then since $maxC(\pi')$ is even, we have that $maxC(\pi') \succcurlyeq^0 g(e)$. If $g(e)$ is even, then $c' = g(e) - 1$, and since $maxC(\pi') > c'$, it must be that $maxC(\pi') \geqslant g(e)$. Since $maxC(\pi')$ is even, it follows that $maxC(\pi') \succcurlyeq^0 g(e)$. Consider now the case where $maxC(\pi) \leqslant maxC(\pi') = c'$. Since $maxC(\pi')$ is even so is c', and thus, it must be that $g(e)$ is odd and that $c' = g(e) + 1$. It follows that either $maxC(\pi) = maxC(\pi')$, in which case $maxC(\pi)$ is even, or that $maxC(\pi) \leqslant g(e)$. Hence, since $g(e)$ is odd, in both cases $maxC(\pi) \succcurlyeq^0 g(e)$.

Assume now that ξ is not losing and achieves g. It follows that all plays consistent with ξ that do not exit $\mathcal{V}_i$ are winning for Player 0. Observe that, by the structure of $\mathcal{V}_{i,g}$, this implies that in order to show that ξ' is winning it is enough to show that no play consistent with ξ' ends in a terminal node of the form $(e,0)$, and that all plays that go through a node of the form $(e,0)$ infinitely often are winning for Player 0. Consider first an exit $e \in exit_i$ such that $(e,0)$ is a terminal node. It follows that $g(e) = \dashv$, and since ξ achieves g no play consistent with ξ can exit $\mathcal{V}_i$ through e. Since e is the only predecessor of $(e,0)$, then no play consistent with ξ' can reach $(e,0)$. Consider now the case where $(e,0)$ is not a terminal node (and thus $g(e) \neq \dashv$). Observe that, by the structure of $\mathcal{V}_{i,g}$, all plays that go infinitely often through $(e,0)$ are a concatenation of infinitely many finite plays from in_i to $(e,0)$. Hence, to complete the proof that ξ' is winning, it is enough to show that for every play $\pi' = in_i \cdot \pi_1 \cdots \pi_k \cdot e \cdot (e,0)$ consistent with ξ' we have that $maxC(\pi')$ is even. Let π' be such a play, and observe that the play $\pi = in_i \cdot \pi_1 \cdots \pi_k \cdot e \cdot out$ is a play consistent with ξ that exits through e. Since, by our assumption, ξ achieves g, then $maxC(\pi) \succcurlyeq^0 g(e)$.

Observe that by the structure of π and π', either $maxC(\pi) = maxC(\pi') > c'$, or $maxC(\pi) \leqslant maxC(\pi') = c'$. Consider first the case where $maxC(\pi) = maxC(\pi') > c'$. If $g(e)$ is even, then since $maxC(\pi) \succcurlyeq^0 g(e)$ we have that $maxC(\pi)$ is also even. If $g(e)$ is odd, then $c' = g(e) + 1$, and thus, $maxC(\pi) > c'$ implies that $maxC(\pi) > g(e)$. Hence, since $maxC(\pi) \succcurlyeq^0 g(e)$ and $g(e)$ is odd, then $maxC(\pi)$ is even. Consider now the case where $maxC(\pi) \leqslant maxC(\pi') = c'$, and assume by way of contradiction that c' is odd. It follows that $g(e)$ is even and that $c' = g(e) - 1$. Since $maxC(\pi) \succcurlyeq^0 g(e)$, and $g(e)$ is even, then $maxC(\pi) \geqslant g(e)$, but this is a contradiction since $maxC(\pi) \leqslant maxC(\pi') = c' = g(e) - 1$. □

Combining Lemma 2 with Theorem 3 we get:

Corollary 1. *A summary function g is relevant iff Player 0 wins $\mathcal{G}^s_{i,g}$.*

Observe that the definition of a summary function of a strategy can also be applied to Player 0 strategies over $\mathcal{V}^s_i$. Since $\mathcal{V}_i$ has the same exit nodes as $\mathcal{V}^s_i$, then the sets of summary functions over $\mathcal{V}_i$ and $\mathcal{V}^s_i$ coincide, and we can compare strategy functions over $\mathcal{V}_i$ with ones over $\mathcal{V}^s_i$ using the relation $\succcurlyeq$. Given a strategy ξ of Player 0 for $\mathcal{V}_i$, we say that a strategy ξ', of Player 0 for $\mathcal{V}^s_i$, is *as good as* ξ, when: (*i*) if ξ is a winning strategy then so is ξ'; and (*ii*) if ξ is not a losing strategy then so is ξ', and $g_{\xi'} \succcurlyeq g_\xi$. We define strategies over $\mathcal{V}_i$ that are as good as strategies over $\mathcal{V}^s_i$ in a symmetric way. We claim that the following holds:

Lemma 3. *For every $1 \leqslant i \leqslant n$, and every memoryless strategy ξ of Player 0 for $\mathcal{V}_i$, there is a memoryless strategy ξ' for $\mathcal{V}^s_i$ that is as good as ξ; and vice versa.*

Before we get to the proof of this lemma, we first need some definitions. Given a memoryless strategy ξ for Player σ over $\mathcal{V}_i$, and a box $b \in \mathcal{B}_i$ that refers to $\mathcal{V}_j$, the *restriction* of ξ to b, denoted by ξ_b, is a memoryless strategy for Player σ over $\mathcal{V}_j$ that is obtained by limiting our attention to nodes inside b, and replacing by *out* every move in ξ that exits b. Formally, let $s = (b_0, \ldots, b_k, w)$ be a node in $\mathcal{V}^f_j$, and observe that $s' = (b, b_0, \ldots, b_k, w)$ is a node of $\mathcal{V}^f_i$. Let $\xi(s') = (b'_0, \ldots, b'_h, w')$, and define $\xi_b(s) = \xi(s')$ if $b'_0 = b$ and there is an edge $(w, w') \in \mathcal{R}_j$ (i.e., if ξ does not move the token from s' outside of b), and define $\xi_b(s) = out$ otherwise. Note that the requirement above that there is an edge $(w, w') \in \mathcal{R}_j$, is to make sure that if the token moved from an exit of $\mathcal{V}_j$ back to its entry by using an edge of $\mathcal{V}_i$ of the form $((b,e),b)$, it would not be wrongly considered as a possible move inside $\mathcal{V}_j$. It is easy to see that ξ_b is indeed a memoryless strategy for Player σ over $\mathcal{V}_j$. Observes that if $b_1, b_2 \in \mathcal{B}_i$ are two different boxes such that $\tau_i(b_1) = \tau_i(b_2) = j$, then it is normally *not* the case that $\xi_{b_1} = \xi_{b_2}$. That is, the choice of how to move inside the sub-arena $\mathcal{V}_j$ may depend on the context in which it appears.

For technical convenience, throughout the proof of Lemma 3, we assume that Player 0 strategies are defined over all Player 0 nodes. This can be easily done by directing missing transitions to a special losing terminal Player 0 node, which we will assume is a node that was added to W_i. Note that if nodes for which the strategy is not defined are not reachable by initial plays, this change makes no semantic difference. Also note that we keep referring to nodes for which the strategy is "undefined", with the understanding that this refers to the state of affairs before the missing transitions are added.

We break the proof to two sub-lemmas, one for each direction.

Lemma 4. *For every $1 \leqslant i \leqslant n$, and every memoryless strategy ξ of Player 0 for $\mathcal{V}_i$, there is a memoryless strategy ξ' for $\mathcal{V}^s_i$ that is as good as ξ.*

Proof. Note that any strategy is as good as a losing strategy. We are thus left with the case that ξ is not losing. Given a non-losing memoryless strategy ξ of Player 0 for $\mathcal{V}_i$, we define a memoryless Player 0 strategy ξ' for $\mathcal{V}_i^s$ as follows. Let s be a Player 0 node of $\mathcal{V}_i^s$, then:

- If $s \in W_i$ then: $\xi'(s) = \xi(s)$ if $\xi(s) \in W_i \cup \{out\}$, and $\xi'(s) = p^b$ if $\xi(s) = (b, in_{\tau_i(b)})$ for some $b \in \mathcal{B}_i$.
- If $s = p^b$ for some $b \in \mathcal{B}_i$, then $\xi'(s) = g^b_{\xi_b}$ if $(b, in_{\tau_i(b)})$ is reachable by some play consistent with ξ, and is otherwise undefined.
- If $s = (e, c)^b$ where $(e, c) \in exit_j \times C$ is a node of some gadget H_j, then: $\xi'(s) = \xi(b, e)$ if $\xi(b, e) \in W_i \cup \{out\}$; $\xi'(s) = p^{b'}$ if $\xi(b, e) = (b', in_{\tau_i(b')})$ for some $b' \in \mathcal{B}_i$; otherwise, $\xi'(s)$ is undefined.

In the second item in the definition above, recall that g_{ξ_b} is the summary function of the restriction of ξ to b. Thus, if $(b, in_{\tau_i(b)})$ is reachable by some play consistent with ξ, then since ξ is not losing it must be that g_{ξ_b} is relevant, and thus, $g^b_{\xi_b}$ is indeed a node of $\mathcal{V}_i^s$. In the third item in the definition above, note that if $\xi(b, e)$ is not of one of the two forms given, then it must be that ξ does not allow the token to exit b through e, in which case (by the way H_j was constructed, and the definition of a summary function) the node $(e, c)^b$ is not reachable from $g^b_{\xi_b}$, which is its only possible predecessor on plays that agree with ξ'. Hence, in such cases we can safely leave $\xi'(s)$ undefined.

We now show that ξ' is as good as ξ. Intuitively, ξ' is as good as ξ iff Player 1 can not do better when playing against ξ' than when playing against ξ. Formally, it is enough to show that given any maximal play π' that is consistent with ξ' and is not losing for Player 1, there is a corresponding maximal play π that is consistent with ξ, such that: (i) π is infinite iff π' is, and if π' is finite then the last node in π is equal to the last node in π' (note that we do not consider the special symbol "*out*" to be a node); (ii) $maxC(\pi) = maxC(\pi')$.

Let $\Omega = \{p^b \cdot g^b_{\xi_b} \cdot (e, g_{\xi_b}(e))^b\colon\ b \in \mathcal{B}_i,\ e \in exit_i,\ g_{\xi_b}(e) \neq \dashv\}$ be the set of all paths consistent with ξ', from roots to leaves of gadgets in $\mathcal{V}_i^s$. Consider first maximal plays π' of the form $(W_i^* + \Omega^*)^\omega + (W_i^* + \Omega^*)^* \cdot exit_i \cdot out$ (we will later see that all maximal plays consistent with ξ', that are not losing for Player 1, are of this form). Given such a play π', we derive from it the required play π, by replacing every sub-word $x' = p^b \cdot g^b_{\xi_b} \cdot (e, g_{\xi_b}(e))^b \in \Omega$ of π' by a word x over $\mathcal{V}_i^{\mathrm{f}}$, as follows. Let y be some play consistent with g_{ξ_b} over the arena $\mathcal{V}^{\mathrm{f}}_{\tau_i(b)}$, that starts in $in_{\tau_i(b)}$ and ends in e, such that the maximal color along y is $g_{\xi_b}(e)$. Observe that by our definition of a summary function such a play exists. The word x is obtained by simply appending b as the first component to each letter of y. I.e., a letter $(u_1, \ldots, u_h)$ in y becomes the letter $(b, u_1, \ldots, u_h)$ in x. Since the only colored node in x' is the node $(e, g_{\xi_b}(e))^b$, and its color is $g_{\xi_b}(e)$, we have that x and x' have the same maximal color, and thus $maxC(\pi) = maxC(\pi')$. It is not hard to see that π is indeed a maximal play consistent with ξ, that it is infinite iff π' is infinite, and that if π' is finite then the last node of π is equal to that of π'.

We now show that every maximal play π' that is consistent with ξ' and is not losing for Player 1 is indeed of the form $(W_i^* + \Omega^*)^\omega + (W_i^* + \Omega^*)^* \cdot exit_i \cdot out$. Note that proving this also establishes that ξ' is well defined, i.e., that no play consistent with it reaches a node for which ξ' is undefined (and thus ends with the special losing terminal node in W_i). By the definition of ξ' and the structure of $\mathcal{V}_i^s$, we only have to show that π' is not of the form $(W_i^* + \Omega^*)^* + (W_i^* + \Omega^*)^* \cdot (\Omega_1 + \Omega_2)$, where Ω_1 and Ω_2 are the sets of prefixes of words in Ω of lengths 1 and 2, respectively. Since by our assumption π' is maximal and not losing for Player 1, the last node of π' must be a Player 0 node. Since all words in Ω_2 end with a Player 1 node, it follows that π' cannot end with a word in Ω_2. Assume now that $\pi' \in (W_i^* + \Omega^*)^*$, and observe that by applying the construction above, which replaces every sub-word $x' \in \Omega$ of π' with a path $x \in \mathcal{V}_i^{\mathrm{f}}$, we derive a play π that is consistent with ξ. Let s, s' be the last nodes of π and π' (respectively), and recall that s' must be a Player 0 node. Observe that if $\pi' \in (W_i^* + \Omega^*)^*$, then $s' = s \in W_i$, or $s' = (e, g_{\xi_b}(e))^b$ and $s = (b, e)$, for some $b \in \mathcal{B}_i$ and $e \in exit_i$. Recall that, by definition, the nodes $(e, g_{\xi_b}(e))^b$ and (b, e) belong to the same player that owns e. It follows that in both cases, s belongs to the player that owns s', and thus it is a Player 0 node. Since we assumed that ξ is not losing (for Player 0), it must be that π can be extended to a longer play, i.e., that $\xi(s)$ is defined. Hence, by the definition of ξ', we also have that $\xi'(s')$ is defined, and thus π' can also be extended, and is not maximal. Finally, to see why π' cannot end with a word in Ω_1 (i.e., with a node of the form p^b), we once more apply the construction that replaces every sub-word $x' \in \Omega$ in π' with a path $x \in \mathcal{V}_i^{\mathrm{f}}$. Furthermore, we replace the last node p^b of π' with $(b, in_{\tau_i(b)})$. We thus obtain a play that is consistent with ξ and reaches $(b, in_{\tau_i(b)})$. By the definition of ξ', it follows that $\xi'(p^b)$ is defined, and thus π' can be extended and is not maximal. □

Lemma 5. *For every $1 \leqslant i \leqslant n$, and every memoryless strategy ξ' of Player 0 for $\mathcal{V}_i^s$, there is a memoryless strategy ξ for $\mathcal{V}_i$ that is as good as ξ'.*

Proof. Note that any strategy is as good as a losing strategy. We are thus left with the case that ξ' is not losing. Consider a non-losing memoryless strategy ξ' of Player 0 for $\mathcal{V}_i^s$. For every box $b \in \mathcal{B}_i$ for which $\xi'(p^b)$ is defined, let g^b be the summary function $g^b = \xi'(p^b)$, and let ϱ^b be some (fixed) arbitrarily chosen memoryless Player 0 strategy for the sub-arena $\mathcal{V}_{\tau_i(b)}$ that achieves g^b. Given $s \in W_i^{0\mathrm{f}}$, we define the strategy ξ as follows:

- If $s \in W_i$ then: $\xi(s) = \xi'(s)$ if $\xi'(s) \in W_i \cup \{out\}$, and $\xi(s) = (b, in_{\tau_i(b)})$ if $\xi'(s) = p^b$ for some $b \in \mathcal{B}_i$.
- If $s = (b, b_1, \ldots, b_k, w)$, where $b \in \mathcal{B}_i$, and $\xi'(p^b)$ is undefined, then $\xi(s)$ is also undefined.
- If $s = (b, b_1, \ldots, b_k, w)$, where $b \in \mathcal{B}_i$, and $\varrho^b(s) \neq out$, then $\xi(s) = \varrho^b(s)$.
- If $s = (b, b_1, \ldots, b_k, w)$, where $b \in \mathcal{B}_i$, and $\varrho^b(s) = out$, then it must be that $s = (b, e)$ where $e \in exit_{\tau_i(b)}$. Let $c = g^b(e)$, then: $\xi(s) = \xi'((e, c)^b)$ if $\xi'((e, c)^b) \in W_i \cup \{out\}$, and $\xi(s) = (b', in_{\tau_i(b')})$ if $\xi'((e, c)^b) = p^{b'}$ for some $b' \in \mathcal{B}_i$.

We now prove that ξ is as good as ξ'. We use a similar argument to the one used in the proof of Lemma 4. Formally, we show that given any maximal play π that is consistent with ξ and is not losing for Player 1, there is a corresponding maximal play π' that is consistent with ξ', such that: (*i*) π' is infinite iff π is, and if π is finite then the last node in π' is equal to the last node in π (note that we do not consider the special symbol "*out*" to be a node); (*ii*) $maxC(\pi) \succcurlyeq_0 maxC(\pi')$.

For every $b \in \mathcal{B}_i$, let $b_{\text{states}} = \{b\} \times W^{\text{f}}_{\tau_i(b)}$ be the set of all states in W^{f}_i whose first coordinate is b, and let $b_{\text{paths}} = \{x \in (b_{\text{states}})^*$: for all $0 \leqslant j < |x|$ we have that $(x_j, x_{j+1}) \in \mathcal{R}^{\text{f}}_i \wedge (x_j \in W^{0^{\text{f}}}_i \Longrightarrow x_{j+1} = \xi(x_j))\}$ be the set of paths inside b that are consistent with ξ. Finally, let $\Upsilon = \bigcup_{b \in \mathcal{B}_i} b_{\text{paths}}$. Consider first maximal plays π of the form $(W^*_i + \Upsilon^*)^\omega + (W^*_i + \Upsilon^*)^* \cdot exit_i \cdot out$ (we will later see that all maximal plays consistent with ξ, that are not losing for Player 1, are of this form). We say that a sub-word $x = \pi_j \cdots \pi_k \in \Upsilon$ of π is *maximal*, iff $(\pi_{j-1} \cdot \pi_j \cdots \pi_k) \notin \Upsilon$ and $(\pi_j \cdots \pi_k \cdot \pi_{k+1}) \notin \Upsilon$. Observe that if $x = x_0 \cdots x_h$ is a maximal sub-word of π, then $x \in b_{\text{paths}}$ for some box $b \in \mathcal{B}_i$, and it represents an entire sequence of moves from the point the token enters b until it exits it. In particular, it must be that $x_0 = (b, in_{\tau_i(b)})$ and that $x_h = (b, e)$ where $e \in exit_{\tau_i(b)}$. Note that π admits a single representation of the form $(W^*_i + \Upsilon^*)^\omega + (W^*_i + \Upsilon^*)^* \cdot exit_i \cdot out$, if sub-words in Υ are chosen to be maximal. Given a play π as above, we derive from it the required play π' by replacing every maximal sub-word $x \in b_{\text{paths}}$ of π by the word $x' = p^b \cdot g^b \cdot (e, g^b(e))^b$. Observe that, by the definition of ξ, the word x represents a play over $\mathcal{V}^{\text{f}}_{\tau_i(b)}$ that is consistent with ϱ^b, and exits through e. Hence, by the definition of a summary function, $maxC(x) \succcurlyeq_0 g_{\varrho^b}(e)$. Recall that ϱ^b achieves g^b and thus $g_{\varrho^b}(e) \succcurlyeq_0 g^b(e)$. From transitivity of $\succcurlyeq_0$ we get that $maxC(x) \succcurlyeq_0 g^b(e)$. Since the only colored node in x' is the node $(e, g^b(e))^b$, and its color is $g^b(e)$, we get that $maxC(x) \succcurlyeq_0 maxC(x')$, and thus overall, $maxC(\pi) \succcurlyeq_0 maxC(\pi')$. It is not hard to see that π' is indeed a maximal play consistent with ξ', that it is infinite iff π is infinite, and that if π is finite then the last nodes of π' and π are the same.

We now show that every maximal play π consistent with ξ, that is not losing for Player 1, is indeed of the form $(W^*_i + \Upsilon^*)^\omega + (W^*_i + \Upsilon^*)^* \cdot exit_i \cdot out$. Note that proving this also establishes that ξ is well defined, i.e., that no play consistent with it reaches a node for which ξ is undefined (and thus ends with the special losing terminal node in W_i). Given $b \in \mathcal{B}_i$, let $b_{\omega\text{-paths}} = \{x \in (b_{\text{states}})^\omega$: for all $0 \leqslant j$ we have that $(x_j, x_{j+1}) \in \mathcal{R}^{\text{f}}_i \wedge (x_j \in W^{0^{\text{f}}}_i \Longrightarrow x_{j+1} = \xi(x_j))\}$ be the set of infinite paths inside b that are consistent with ξ, and let $\Upsilon_\omega = \bigcup_{b \in \mathcal{B}_i} b_{\omega\text{-paths}}$. Note that by the definition of ξ, and the structure of $\mathcal{V}_i$, we only have to show that π is not of the form $(W^*_i + \Upsilon^*)^* \cdot W_i + (W^*_i + \Upsilon^*)^* \cdot (\Upsilon + \Upsilon_\omega)$, i.e., that π cannot reach a terminal node in W_i, or never come out of a nested sub-arena. Assume first that $\pi \in (W^*_i + \Upsilon^*)^* \cdot W_i$. Since by our assumption π is maximal and not losing for Player 1, the last node $s \in W_i$ of π must be a Player 0 node. Observe that by applying the construction above, that replaces every maximal sub-word $x \in \Upsilon$ of π with a 3-node path $x' \in \mathcal{V}'_i$, we derive a play π' that is consistent with ξ' and ends with s. Since we assumed that ξ' is not a losing strategy, and we know that s is a Player 0 node, it must be that π' can be extended to a longer play, i.e., that $\xi'(s)$ is defined. Hence, by the definition of ξ, we also have that $\xi(s)$ is defined, and thus π can also be extended, and is not maximal. Assume now that $\pi \in (W^*_i + \Upsilon^*)^* \cdot (\Upsilon + \Upsilon_\omega)$, and let $\pi_0 \cdots \pi_k$ be the shortest prefix of π such that the suffix $\pi_{k+1} \cdots$ is in $\Upsilon + \Upsilon_\omega$. It follows that there is a box $b \in \mathcal{B}_i$ such that $\pi_{k+1} = (b, in_{\tau_i(b)})$ and for every $j \geqslant k$ we have that $\pi_j \in b_{\text{states}}$. Furthermore, by the definition of ξ, there are only two options: (*i*) $\xi'(p^b) = g^b$, and the suffix $\pi_{k+1} \cdots$ of π is a play consistent with ϱ^b over the sub-arena $\mathcal{V}^{\text{f}}_{\tau_i(b)}$; or (*ii*) $\xi'(p^b)$ is undefined. To see why the first option is impossible, observe that since ϱ^b is not a losing strategy (it achieves the relevant summary function g^b), the suffix $\pi_{k+1} \cdots$, and hence also π, is not losing for Player 0. On the other hand, by our assumption, π is not losing for Player 1, which is a contradiction (recall that a play that does not end with *out* cannot be a tie). To see why the second option is also impossible, we apply to the prefix $\pi_0 \cdots \pi_k$ the construction above that replaces maximal sub-words $x \in \Upsilon$ with 3-node paths $x' \in \mathcal{V}'_i$. We thus derive a play π' over $\mathcal{V}^{\text{s}}_i$ such that $\pi' \cdot p^b$ is consistent with ξ'. Since by our assumption ξ' is not a losing strategy, the play $\pi' \cdot p^b$ cannot be losing for Player 0, and thus, since p^b is a Player 0 node, $\xi'(p^b)$ must be defined. □

Recall that $\mathcal{V}^{\text{s}} = \mathcal{V}^{\text{s}}_{1,\varepsilon}$, and thus, by applying Lemma 3 to the arenas $\mathcal{V}_1$ and $\mathcal{V}^{\text{s}}_1$, we get that:

Theorem 3. *Given a hierarchical parity game $\mathcal{G} = (\mathcal{V}, \Gamma)$, Player 0 wins the game iff he wins the simple parity game $\mathcal{G}^{\text{s}}_{1,\varepsilon} = (\mathcal{V}^{\text{s}}_{1,\varepsilon}, \Gamma^{\text{s}}_{1,\varepsilon})$.*

Analyzing the time and space requirements of our algorithm for solving hierarchical parity games, we get the following:

Theorem 4. *Let $\mathcal{G} = (\mathcal{V}, \Gamma)$ be a hierarchical parity game with $|C|$ colors, and $e = exits(\mathcal{V})$. Solving $\mathcal{G}$ can be done in time $2^{(|C| \cdot \log |\mathcal{V}| + O(|C| \cdot e \cdot \log |C|))}$, and it is* PSPACE*-complete.*

Proof. We start with the time complexity. It is not hard to see that the runtime of the algorithm is dominated by the time spent in deciding which summary functions are relevant. For every $1 \leqslant i \leqslant n$, and every summary function $g \in Summ(\mathcal{V}_i)$, the algorithm has to solve one simple parity game $\mathcal{G}^s_{i,g} = (\mathcal{V}^s_{i,g}, \Gamma^s_{i,g})$. The number of summary functions is $|Summ(\mathcal{V}_i)| = (|C|+1)^{|exit_i|}$, and thus, the number of nodes in the gadget H_i is $O((|C|+1)^{|exit_i|})$. Hence, the size of the arena $\mathcal{V}^s_{i,g}$ is $S_i = |\mathcal{V}_i| + \sum_{b \in \mathcal{B}_i} O((|C|+1)^{|exit_{\tau_i(b)}|}) = |\mathcal{V}_i| \cdot O((|C|+1)^{exits(\mathcal{V})})$. Overall, for every $1 \leqslant i \leqslant n$, the algorithm solves at most $(|C|+1)^{exits(\mathcal{V})}$ such simple parity games. By [36], every such game can be solved in time $O(S_i^{|C|})$.[8] Therefore, the overall time complexity is $|\mathcal{V}|^{|C|} \cdot |C|^{O(|C| \cdot exits(\mathcal{V}))}$.

It is interesting to note that if the number of exits of $\mathcal{V}$ is poly-logarithmic in $|\mathcal{V}|$ (and in particular if it is constant), then the number of arenas as well as their sizes is polynomial in $|\mathcal{V}|$. Thus, solving hierarchical parity games of this type is not harder then solving simple parity games.

We now proceed to analyze the space complexity, and show that our algorithm can be implemented in space $O(nd(\mathcal{V}) \cdot (|\mathcal{V}| \cdot exits(\mathcal{V}) \cdot \log|C| + |\mathcal{V}| \cdot \log|\mathcal{V}|))$, where $nd(\mathcal{V})$ is the nesting depth of $\mathcal{V}$. For every $1 \leqslant i \leqslant n$, and every summary function $g \in Summ(\mathcal{V}_i)$, consider the simple parity game $\mathcal{G}^s_{i,g}$ that the algorithm has to solve. Note that here we can not use just any parity solver for $\mathcal{G}^s_{i,g}$, and we have to use one which is space efficient. The key observation is that the cause of the exponential blow-up in the number of nodes of $\mathcal{V}^s_{i,g}$, compared to the hierarchical sub-arena $\mathcal{V}_i$, is the set of nodes $\{g^b\colon\ b \in \mathcal{B}_i \wedge g \in \mathcal{M}_{\tau_i(b)}\}$ (i.e., the nodes of summary functions found in the different gadgets inside $\mathcal{V}^s_{i,g}$), and that all nodes in this set are Player 1 nodes. Hence, the space required to remember a memoryless strategy of Player 0 for $\mathcal{V}^s_{i,g}$ is polynomial in $|\mathcal{V}_i|$, and not exponential. Let E be the set of edges of $\mathcal{V}^s_{i,g}$. Let D be the set of its nodes, let $P \subseteq D$ be the set of its Player 0 nodes, and let $P^{\mathcal{B}} = \{p^b \in P\colon\ b \in \mathcal{B}_i\}$ be the set of all nodes in $\mathcal{V}^s_{i,g}$ that are entries (root nodes) of gadgets. Recall that a memoryless strategy ξ of Player 0 for $\mathcal{V}^s_{i,g}$ is a function $\xi : P \to D$. Note, however, that it can also be viewed as a pair of functions $\xi = (\dot{\xi}, \ddot{\xi})$ where $\dot{\xi} : P \setminus P^{\mathcal{B}} \to D$, and $\ddot{\xi} : P^{\mathcal{B}} \to \bigcup_{j=i+1}^{n} M_j$, where M_j is the set of all relevant summary functions for $\mathcal{V}_j$. We can over-approximate the set of memoryless strategies of Player 0 by considering functions $\ddot{\xi} : P^{\mathcal{B}} \to \bigcup_{j=i+1}^{n} Summ(\mathcal{V}_j)$, that may assign to a node $p^b \in P^{\mathcal{B}}$ a successor which is a strategy function that is not necessarily relevant. Given such an over-approximation $\xi = (\dot{\xi}, \ddot{\xi})$, let $G_\xi = (V_\xi, E_\xi)$ be the graph induced by ξ, i.e., $V_\xi = \xi(P) \cup (D \setminus succ(P))$, where $succ(P) = \{v \in D\colon\ (u,v) \in E \text{ only if } u \in P\}$ is the set of successors of nodes exclusively in P, and $E_\xi = \{(u,v) \in V_\xi \times V_\xi\colon\ (u,v) \in E\}$. Note that since $succ(P)$ contains all the summary function nodes of all the gadgets in $\mathcal{V}^s_{i,g}$, the number of nodes in V_ξ (that are reachable from ini_i) is $O(|\mathcal{V}_i|)$. Also, note that if ξ is an actual memoryless strategy of Player 0 (i.e., if for all $p^b \in P$ we have that $\ddot{\xi}(p^b) \in \bigcup_{j=i+1}^{n} M_j$), then G_ξ is a subgraph of $\mathcal{V}^s_{i,g}$, and it contains all the possible moves for Player 1.

This leads us to the following space efficient procedure `solve`(i,g) for solving the simple parity game over $\mathcal{V}^s_{i,g}$. The procedure goes (lexicographically) over all possible over-approximations ξ of memoryless strategies of Player 0 for $\mathcal{V}^s_{i,g}$. For each such over-approximation, the procedure checks if the graph G_ξ contains a reachable cycle with a maximal color that is odd (this is the classic procedure used to check if a memoryless strategy of a simple parity game is losing). If there is such a cycle, the procedure goes on to try the next over-approximating strategy; otherwise, it checks to see if ξ is a real strategy or a superfluous over-approximation, by checking for every $p^b \in P^{\mathcal{B}}$, whether the summary function $\xi(p^b)$ is relevant. This check is done by a recursive call to `solve`$(\tau_i(b), \xi(p^b))$. The procedure `solve(i,g)` needs to remember the currently guessed strategy ξ, which requires space $O(|\mathcal{V}| \cdot \log|\mathcal{V}|)$ for $\dot{\xi}$, and $O(|\mathcal{B}_i| \cdot exits(\mathcal{V}) \cdot \log|C|)$ for $\ddot{\xi}$. In addition, the memory required for the cycle-detection phase over the graph G_ξ, is $O(\log^2|\mathcal{V}|)$. Since the depth of the recursive calls to `solve()` is at most the nesting depth of the hierarchical system, we get that solving the game $\mathcal{G}^s_{1,\varepsilon}$ can be done in space $O(nd(\mathcal{V}) \cdot (|\mathcal{V}| \cdot exits(\mathcal{V}) \cdot \log|C| + |\mathcal{V}| \cdot \log|\mathcal{V}|))$. □

We conclude this section with a theorem that specifies the model-checking complexity for various branching-time temporal logics. Given a hierarchical system $\mathcal{K}$ and a branching-time temporal-logic formula φ, the time complexity of model checking $\mathcal{K}$ with respect to φ follows by applying our algorithm for solving hierarchical parity games to the game $\mathcal{G}_{\mathcal{K},\mathcal{A}_\varphi} = (\mathcal{V}, \Gamma)$, where A_φ is an APT accepting exactly the set of trees satisfying the formula φ. In particular, we recall that by Theorem 1, if φ is a CTL or an alternation-free μ-calculus formula, then $\mathcal{A}_\varphi$ has $O(|\varphi|)$ states and index 2, if φ is a CTL* formula, then $\mathcal{A}_\varphi$ has $2^{O(|\varphi|)}$ states and index 3, and if φ is a μ-calculus formula, then $\mathcal{A}_\varphi$ has $O(|\varphi|)$ states and index $O(|\varphi|)$. Let h be the number of states of $\mathcal{A}_\varphi$, observe that $|\mathcal{V}| = |\mathcal{K}| \cdot h$, $exits(\mathcal{V}) = exits(\mathcal{K}) \cdot h$ and the number of sub-arenas of $\mathcal{V}$ is h times the number of sub-structures of $\mathcal{K}$. As we show in Theorem 4 our algorithm for solving hierarchical parity games can be implemented in polynomial space, which gives an alternative proof of the PSPACE upper bound for the hierarchical μ-calculus model checking given in [18]. For the other logics, a PSPACE upper bound follows by simply flattening the system and applying the LOGSPACE algorithm from [23]. The PSPACE lower-bound for all these logics follows from the known result about CTL [6]. Note that for the logic CTL, the time complexity of the model-checking problem was already

[8] Better complexities are known, but for the sake of simplicity we use the $O(S_i^{|C|})$ bound, and do not bother tightening the complexity here.

Table 1
Model checking complexity results.

	Kripke structure	Hierarchical structure	Pushdown system
μ-calculus	UP ∩ co-UP [36] $O((\lvert\mathcal{K}\rvert \cdot \lvert\varphi\rvert)^{\frac{l+1}{2}})$	PSPACE-comp. $(\lvert\mathcal{K}\rvert \cdot \lvert\varphi\rvert)^l \cdot 2^{O(\lvert\varphi\rvert)\cdot e\cdot l\cdot \log l}$	EXPTIME-comp. [7,32] $2^{O(\lvert\mathcal{K}\rvert\cdot\lvert\varphi\rvert\cdot l^2)}$
alternation free μ-calculus	linear-time [11] $O(\lvert\mathcal{K}\rvert \cdot \varphi)$	PSPACE-comp. $2^{(2\log\lvert\mathcal{K}\rvert + O(\lvert\varphi\rvert)\cdot e)}$	EXPTIME-comp. [7,32] $2^{O(\lvert\mathcal{K}\rvert\cdot\lvert\varphi\rvert)}$
CTL*	PSPACE-comp. [16] $\lvert K\rvert \cdot 2^{O(\lvert\varphi\rvert)}$	PSPACE-comp. $2^{(3\log\lvert\mathcal{K}\rvert + 2^{O(\lvert\varphi\rvert)}\cdot e)}$	2EXPTIME-comp. [8] $2^{(\lvert\mathcal{K}\rvert 2^{O(\lvert\varphi\rvert)})}$
CTL	linear-time [9] $O(\lvert\mathcal{K}\rvert \cdot \varphi)$	PSPACE-comp. $2^{(2\log\lvert\mathcal{K}\rvert + O(\lvert\varphi\rvert)\cdot e)}$	EXPTIME-comp. [8,33] $2^{O(\lvert\mathcal{K}\rvert\cdot\lvert\varphi\rvert)}$

known and our algorithm suggests an alternative to the one in [6]. For the other logics, our approach leads to improved time complexities.

Theorem 5. *Consider a hierarchical system $\mathcal{K}$ and a specification φ for it. Let e be the number of exits of the system, and l be the alternation depth of φ.*

- *For the μ-calculus, the model-checking problem is PSPACE-complete and can be solved in time $(\lvert\mathcal{K}\rvert \cdot \lvert\varphi\rvert)^l \cdot 2^{O(\lvert\varphi\rvert)\cdot e\cdot l\cdot\log l}$.*
- *For CTL and the alternation-free μ-calculus, the model-checking problem is PSPACE-complete and can be solved in time $2^{(2\log\lvert\mathcal{K}\rvert + O(\lvert\varphi\rvert)\cdot e)}$.*
- *For CTL*, the model-checking problem is PSPACE-complete and can be solved in time $2^{(3\log\lvert\mathcal{K}\rvert + 2^{O(\lvert\varphi\rvert)}\cdot e)}$.*

Table 1 shows our results in comparison to the complexity of model-checking of standard Kripke structures on the one hand, and pushdown systems on the other hand. In the table, e denotes the number of exits of the system, and l is the alternation depth of the formula. Note that for all branching-time temporal logics we consider, the hierarchical setting is easier than the setting of pushdown systems.

5. An abstraction-refinement paradigm

In [31], Shoham and Grumberg defined 3-valued games and used them to describe an abstraction-refinement framework for CTL. In this section, we lift their contribution to hierarchical systems. As we show, the idea of summary functions can be applied also for solving *hierarchical* 3*-valued games*. We first define hierarchical 3-valued games, hierarchical modal transition systems, and the semantics of CTL with respect to them. Once the notions are defined, combining the algorithm in Section 4 for the concrete hierarchical setting and the game-based approach to abstraction-refinement for the flat setting [31] into a game-based approach to abstraction-refinement of hierarchical systems is not technically difficult. Essentially, the idea is as follows. In a 2-valued game, the goal of a player is to win. In a 3-valued game, the goal of a player is to win or (in case he cannot win) not to lose (that is, force the game to an "unknown" winning value). Accordingly, the lifting of the algorithm in Section 4 to the 3-valued setting is based on adding a layer to the gadgets H_i described there; a layer in which Player 0 chooses between winning and not losing.

As in the flat setting, abstraction is based on merging sets of states of the concrete system into abstract states. What makes the hierarchical setting interesting is the fact that now it is possible to merge also boxes. Consider a (concrete) hierarchical structure. A sub-structure typically stands for a function, and a call to a function g from within another function f is modeled by a box inside the sub-structure modeling f that refers to the sub-structure modeling g. The values of the local variables of f are typically different in different calls to g. Thus, the source of complexity is not the number of sub-structures, and rather it is the number of states and boxes in each sub-structure. Accordingly, our abstraction does not try to merge sub-systems and contains one abstract sub-system for each concrete sub-system. Our abstraction does merge sets of concrete states into a single abstract state and sets of concrete boxes (referring to the same structures) into a single abstract box.

A *hierarchical* 3*-valued game* is similar to a hierarchical game, only that there are two transition relations $Rmust_i$ and $Rmay_i$, referred to as the *must* and *may* transitions. The transitions are defined as $\mathcal{R}_i$ in a hierarchical game and satisfy $Rmust_i \subseteq Rmay_i$. A *hierarchical modal transition system* (HMTS) over AP is then similar to a hierarchical system, only that, again, there are both must and may transitions, and the labeling function $\sigma_i : W_i \times AP \to \{tt, ff, \bot\}$ can map an atomic proposition also to $\bot$ (unknown). Note that, equivalently, we could have defined HMTS by adding hierarchy to the MTS of [25].

Given a (concrete) hierarchical system $\mathcal{K} = \langle \mathcal{K}_1, \ldots, \mathcal{K}_n \rangle$, with $\mathcal{K}_i = \langle AP, W_i, \mathcal{B}_i, in_i, exit_i, \tau_i, \mathcal{R}_i, \sigma_i \rangle$, an abstraction of $\mathcal{K}$ is an HMTS $\mathcal{M} = \langle \mathcal{M}_1^A, \ldots, \mathcal{M}_n^A \rangle$, where for every $1 \leqslant i \leqslant n$, the sub-model $\mathcal{M}_i^A = \langle AP, W_i^A, \mathcal{B}_i^A, in_i^A, exit_i^A, \tau_i^A, Rmust_i, Rmay_i, \sigma_i^A \rangle$ of $\mathcal{M}$ is an abstraction of the sub-structure $\mathcal{K}_i$, defined as follows. The set of abstract states is $W_i^A \subseteq 2^{W_i}$, and

it forms a partition of W_i. The set of abstract boxes is $\mathcal{B}_i^A$, it forms a partition of $\mathcal{B}_i$, and an abstract box contains only concrete boxes that refer to the same sub-structure. Thus, if $b, b' \in b^a \in \mathcal{B}_i^A$, then $\tau_i(b) = \tau_i(b')$. The latter guarantees that the indexing function $\tau_i^A : \mathcal{B}_i^A \to \{i+1, \ldots, n\}$, defined by $\tau_i^A(b^a) = \tau_i(b)$, for some $b \in b^a$, is well defined. The initial state in_i^A is such that $in_i \in in_i^A$. The set of abstract exits $exit_i^A \subseteq W_i^A$ is such that $e^a \in exit_i^A$ iff $e^a \cap exit_i \neq \emptyset$. Thus, the abstract initial state contains the concrete initial state, and an abstract exit contains at least one concrete exit. The transition relations $Rmay_i$ and $Rmust_i$ are subsets of $(\bigcup_{b \in \mathcal{B}_i^A}(\{b\} \times exit_{\tau_i^A(b)}) \cup W_i^A) \times (W_i^A \cup \mathcal{B}_i^A)$, and are over- and under-approximations of the concrete transitions. Given $w^a = (b^a, e^a) \in \bigcup_{b^a \in \mathcal{B}_i^A}(\{b^a\} \times exit_{\tau_i^A(b^a)})$, we write $w_c \in w^a$ if $w_c = (b_c, e_c)$, $b_c \in b^a$, and $e_c \in e^a$. Using the above notation, we have that $(w^a, w^{a'}) \in Rmay_i$ if there exist $w_c \in w^a$ and $w'_c \in w^{a'}$ such that $(w_c, w'_c) \in R_i$; and $(w^a, w^{a'}) \in Rmust_i$ only if for all $w_c \in w^a$ there exists $w'_c \in w^{a'}$ such that $(w_c, w'_c) \in R_i$. Finally, an atomic proposition holds (does not hold) in an abstract state if it holds (does not hold) in all the concrete states in it; otherwise, its truth value is undefined.

As shown for hierarchical systems, an HMTS $\mathcal{M}$ can be translated to a flat modal transition system (MTS) $\mathcal{M}^{\mathrm{f}}$ by means of the flattening operation (since we only consider abstractions in which all the concrete boxes in an abstract box refer to the same structure, the flattening described for concrete systems can indeed be applied). The semantics of a temporal-logic formula φ over $\mathcal{M}$ is thus simply defined to be the semantics of φ over $\mathcal{M}^{\mathrm{f}}$. For the latter, we use the 3-valued semantics introduced in [21]. The idea is that since may transitions over-approximate concrete transitions, they are used to verify universal formulas or to refute existential formulas. Dually, since must transitions under-approximate concrete transitions, they are used to verify existential formulas or to refute universal formulas. We use $[\mathcal{M}^A \models \varphi]$ to denote the truth value (in $\{tt, ff, \bot\}$) of φ in $\mathcal{M}^A$. Applying the same considerations applied to MTSs [17], it is not hard to see that if an HMTS $\mathcal{M}^A$ abstracts a hierarchical structure $\mathcal{K}$, then $[\mathcal{M}^A \models \varphi] = tt(ff)$ implies that $\mathcal{K} \models \varphi$ (resp. $\mathcal{K} \not\models \varphi$).

Given an HMTS $\mathcal{M}$, and a CTL formula φ, we reduce the problem of deciding the value of $[\mathcal{M}^A \models \varphi]$, to solving a 3-valued game $\mathcal{G}_{\mathcal{M},\mathcal{A}_\varphi}$ obtained by taking the product of $\mathcal{M}$ with the weak alternating tree automaton $\mathcal{A}_\varphi$. The reason we restrict attention to CTL formulas is that taking the product of an HMTS with a weak automaton that corresponds to a CTL formula, there is a distinction between information lost in $\mathcal{M}$ due to atomic propositions whose value is unknown and information lost due to may and must transitions. Indeed, the states of the weak automaton are associated with either atomic propositions (in which case only the first type of missed information should be taken into an account) or with a subformula of the form AX or EX (where only the second type should be taken into an account). Furthermore, in the second case, the game is in either a universal (AX) or existential (EX) mode, so players can proceed along the must and may transitions in their attempt to prove or refute φ. In [19], the authors lifted the abstraction-refinement framework of [31] to handle specifications in the μ-calculus. This suggests that with further technical effort, our algorithm here can also be lifted to the μ-calculus too. We leave this for future research.

Now, as in [31], both players try to either prove or refute φ, and winning strategies must be *consistent*: all transitions taken during a play are must transitions (note that the consistency requirement applies only to winning strategies; the opponent can take also may transitions). Also, a winning strategy cannot end in a state associated with an atomic proposition whose value is unknown. It may be that none of the players have a winning strategy, in which case the value of the game is $\bot$. As described in Section 3 for concrete systems, the hierarchy in the system induces the hierarchy in the product game.

We define $\mathcal{G}_{\mathcal{M},\mathcal{A}_\varphi} = (\mathcal{V}, \Gamma)$ as follows. The arena $\mathcal{V}$ is different from that of the concrete games considered in Section 3 in two aspects. First, since $\mathcal{M}$ has both may and must transitions then so does $\mathcal{V}$. Second, since whether or not an atomic proposition holds at a state of $\mathcal{M}$ may be unknown, and the moves of the automaton $\mathcal{A}_\varphi$ depend on this information, we have to define the transitions of the arena accordingly. Formally, given an HMTS $\mathcal{M} = \langle \mathcal{M}_1, \ldots, \mathcal{M}_n \rangle$ and an APT $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, the hierarchical two-player game $\mathcal{G}_{\mathcal{M},\mathcal{A}} = (\mathcal{V}, \Gamma)$ for $\mathcal{M}$ and $\mathcal{A}$ is defined as follows. The hierarchical arena $\mathcal{V}$ has a sub-arena $\mathcal{V}_{i,q}$ for every $2 \leqslant i \leqslant n$ and state $q \in Q$. For $i = 1$, we need only the sub-arena V_{1,q_0}. The hierarchical order of the sub-arenas is consistent with the one in $\mathcal{K}$. Thus, the sub-arena $\mathcal{V}_{i,q}$ can be referred to by boxes of sub-arena $\mathcal{V}_{j,p}$ only if $i > j$. Let $\mathcal{M}_i^A = \langle AP, W_i^A, \mathcal{B}_i^A, in_i^A, exit_i^A, \tau_i^A, Rmust'_i, Rmay'_i, \sigma_i^A \rangle$ and let $\mathcal{A} = \langle 2^{AP}, Q, q_0, \delta, F \rangle$ be an APT with Q partitioned to $Q^{(\varepsilon,\wedge)}$, $Q^{(\varepsilon,\vee)}$, $Q^{\wedge}$, and $Q^{\vee}$. The sub-arena $\mathcal{V}_{i,q} = \langle W_i^0, W_i^1, \mathcal{B}_i, in_i, exit_i, \tau_i, Rmust_i, Rmay_i \rangle$ is defined as follows.

- $W_i^0 = W_i^A \times (Q^{\vee} \cup Q^{(\varepsilon,\vee)}) \cup \{\dashv\}$, $W_i^1 = W_i^A \times (Q^{\wedge} \cup Q^{(\varepsilon,\wedge)})$, $in_i = (in_i^A, q)$, and $exit_i = exit_i^A \times Q$.
- $\mathcal{B}_i = \mathcal{B}_i^A \times Q$, and $\tau_i((b, q)) = (\tau_i^A(b), q)$.
- For $\mathcal{R}x \in \{Rmust_i, Rmay_i\}$, the relation $\mathcal{R}x$ contains all pairs (u, v) that satisfy the following. Let $u = (w, q)$ or $u = ((b, q''), (w, q))$.
  1. If $\delta(q, \{p \in AP\colon\ \sigma_i^A(w, p) = tt\}) \neq \delta(q, \{p \in AP\colon\ \sigma_i^A(w, p) \neq ff\})$, then $v = \bot$;
  2. Otherwise, if $q \in Q^{\varepsilon}$ and $\delta(q, \sigma_i^A(w)) = (p_0, p_1)$, then $v \in \{(w, p_0), (w, p_1)\}$;
  3. Otherwise, if $q \in Q^{\vee,\wedge}$, then $v = (w', \delta(q, \sigma_i^A(w)))$ and either $(w, w') \in \mathcal{R}x$, if $u = (w, q)$, or $((b, w), w') \in \mathcal{R}x$, otherwise.

Note that the definition above is simply a technical merge of the construction of Section 3, and the one in [17].

As for concrete systems, the coloring of states is induced by the acceptance condition of the automaton, i.e., for each state (w, q) of a sub-arena $\mathcal{V}_{i,q}$, we have $\Gamma(w, q) = F(q)$. However, in order to accommodate the possibility that $[\mathcal{M}^A \models \varphi] = \bot$,

we need to modify winning condition. Intuitively, the players use must transitions in order to win the game and may transitions in order to prevent the other player from winning. As a result it is possible that none of the players wins the play, i.e. the play ends with a tie. Formally, a play is winning for Player 0 if it ends in a terminal node that belongs to Player 1; or if the play is infinite and satisfies Γ. Similarly, a play is winning for Player 1 if it ends in a terminal node that belongs to Player 0 (other than $\perp$), or if the play is infinite and does not satisfy the winning condition Γ. In all other cases the play is a tie. In light of the above, it is not hard to see that we have the following theorem:

Theorem 6. *Given an HMTS $\mathcal{M}$ and a* CTL *formula φ, let $\mathcal{G}_{\mathcal{M},\mathcal{A}_\varphi}$ be the product of $\mathcal{M}$ with $\mathcal{A}_\varphi$. Then*:

- *Player* 0 *has a winning strategy in $\mathcal{G}_{\mathcal{M},\mathcal{A}_\varphi}$ iff $[\mathcal{M} \models \varphi] = tt$.*
- *Player* 1 *has a winning strategy in $\mathcal{G}_{\mathcal{M},\mathcal{A}_\varphi}$ iff $[\mathcal{M} \models \varphi] = ff$.*
- *None of the players have a winning strategy in $\mathcal{G}_{\mathcal{M},\mathcal{A}_\varphi}$ iff $[\mathcal{M} \models \varphi] = \perp$.*

5.1. Solving the game $\mathcal{G}_{\mathcal{M},\mathcal{A}_\varphi}$

The game $\mathcal{G}_{\mathcal{M},\mathcal{A}_\varphi}$ can be solved by adapting the algorithm defined in Section 4 in the following way. Recall that in a game played over a concrete arena each player has only one goal: to try and win. On the other hand, since a play over an abstract arena may be a tie, a player may either try to win, in which case it only uses must transitions, or it may try not to lose, in which case it can also use may transitions. Consider a strategy ξ of Player 0 for an abstract sub-arena $\mathcal{V}_i$, to fully capture the possible responses of Player 1 to ξ, we have to associate with ξ two summary functions: g_ξ^{must} and g_ξ^{may}. The function g_ξ^{must} captures the possible responses of Player 1 if it only uses must transitions (i.e., it tries to win), while g_ξ^{may} captures the possible responses of Player 1 if it uses may transitions (i.e., it tries not to lose). Note that whether Player 0 uses only must transitions or not, is specified by ξ. For every $x = (x_0, x_1) \in \{may, must\} \times \{may, must\}$ we say that a summary function g is *x-feasible* (*x-relevant*) if it is feasible (relevant) when Player 0 uses only x_0 transitions, and Player 1 uses only x_1 transitions. It is easy to see that by limiting attention to only the specified types of transitions, the algorithm presented earlier for deciding whether a summary function (over a concrete arena) is relevant can be used to decide if a summary function is x-relevant. For every $x \in \{may, must\} \times \{may, must\}$, let M_j^x be the set of all x-relevant summary functions for $\mathcal{V}_j$. Observe that $M_j^x \subseteq M_j^y$ if y was obtained from x by changing a must to a may.

In order to reflect the players' choice whether or not to use only must transitions, we adjust the construction of the gadget H_j, which is used to replace a sub-arena $\mathcal{V}_j$, to produce the following 4-level DAG structure:

- Its set of nodes is
$$\{p, t^{may}, t^{must}\} \cup M_j^{(may,may)} \cup (exit_j \times C).$$
- The node p is a Player 1 node, t^{may} and t^{must} are Player 0 nodes, every $g \in M_j^{(may,may)}$ is a Player 1 node, and a node $(e, c) \in exit_j \times C$ belongs to the same player that e belongs to.
- Its set of may edges is
$$\{(p, t^{may})\} \bigcup_{g \in M_j^{(may,may)}} \{(t^{may}, g)\} \bigcup_{g \in M_j^{(may,must)}} \{(t^{must}, g)\}.$$
- Its set of must edges is
$$\{(p, t^{must})\} \bigcup_{g \in M_j^{(must,may)}} \{(t^{may}, g)\} \bigcup_{g \in M_j^{(must,must)}} \{(t^{must}, g)\} \bigcup_{g \in M_j^{(may,may)}} \{(g, (e, g(e))) \colon e \in exit_j \wedge g(e) \neq \dashv\}.$$
- A node $(e, c) \in exit_j \times C$ is colored by c. These are the only colored nodes.

Intuitively, at the entrance p Player 1 makes the choice whether he wants to only use must transitions, in which case he takes the must transition to t^{must}, or to use may transitions, in which case he takes the may transition to t^{may}. Note that if Player 1 has a winning strategy using only must transitions (and Player 0 is not limited) he would surely use it; otherwise, Player 0 either has a winning strategy or it can force a tie, and thus Player 1 can only lose by limiting itself to must transitions, and it would decide to use may transitions. Since this line of reasoning is independent of the specific strategy that Player 0 may choose, we are justified in assuming that Player 1 makes this choice upfront. From the node t^{must}, Player 0 chooses a summary function node g that reflects its strategy for the sub-arena $\mathcal{V}_j$. If g is $(must, must)$-relevant then this transition is a must transition, reflecting the fact that Player 0 can achieve g using only must transition for his moves inside the sub-arena $\mathcal{V}_j$; otherwise, g is only $(may, must)$-relevant, and the transition is a may transition that is not a must transition. Observe that there are no edges from t^{must} to g if g is not $(may, must)$-relevant (and is only (may, may)-relevant), since moves from t^{must} must reflect the fact that Player 1 chooses to limit itself to must transitions inside $\mathcal{V}_j$. The possible moves from the node t^{may} follow the same reasoning. Finally, as for concrete games, Player 1 can move from

a node g to any node $(e, g(e))$ for which $g(e) \neq \dashv$. Note that all such transitions are must transitions since both players' choices whether or not to use may transitions are already reflected by the preceding moves (from p to t^{must} or t^{may}, and from there to g).

A very important feature of our construction above is that the gadgets used to replace boxes bare a direct and very natural connection with the abstract hierarchical system that is being model checked. To see this connection, consider for example the case of model checking a CTL formula φ. The states of the automaton $\mathcal{A}_\varphi$ are sub-formulas of φ, and the nodes of a sub-arena $\mathcal{V}_{i,q}$ of the membership game are pairs consisting of a state $s \in W_i^A$ of the abstract HMTS, and a sub-formula ψ. The node (s, ψ) is an exit of the sub-arena iff s is an exit of the sub-structure $\mathcal{V}_i^A$. Intuitively, when the token is placed on the node (s, ψ), Player 0 (Player 1) has a winning strategy iff it can prove that ψ holds (does not hold) at the state s of the sub-structure (in the current context). Observe that for CTL we need just two colors, $\{1, 2\}$, and thus a summary function g over $\mathcal{V}_{i,\psi}$ assigns to every exit pair (e, ψ'), where e is an exit of $\mathcal{V}_i^A$, and ψ' a sub-formula in the closure of ψ, a value in $\{1, 2, \dashv\}$. When Player 0 moves the token to a node g in the gadget that replaces a box that refers to the sub-arena $\mathcal{V}_{i,\psi}$, it essentially claims that it can prove that ψ holds at the initial state of the sub-structure $\mathcal{V}_i$, and that this proof depends on certain assumptions as to which sub-formulas of ψ hold at which exit of $\mathcal{V}_{i,\psi}$ (which depends on the context of this reference to $\mathcal{V}_{i,\psi}$) as specified by the context function g. The fact that we limit the gadgets to use only relevant summary functions means that Player 0 is only allowed to move to g if it can indeed prove that ψ holds at the initial state of the sub-structure $\mathcal{V}_i$ under these assumptions. If $g(e, \psi') = \dashv$, it means that Player 0 makes no assumptions as to whether or not ψ' holds at e; if $g(e, \psi') = 2$, it means that Player 0 assumes that ψ' holds at e; and if $g(e, \psi') = 1$, it means that Player 0 assumes that ψ' holds at e, and that he must prove it without forming a cycle that re-enters the gadget (the case of $g(e, \psi') = 1$ is only possible if $\psi = \psi' = \theta U \theta'$ is an until formula, and such a cycle corresponds to trying to delay the forever the satisfaction of θ'). When Player 1 moves the token from the node g, to $((e, \psi'), g(e, \psi'))$, it has the intuitive meaning that it wants Player 0 to make good on his word and actually prove that in the current context ψ' holds at the exit e of $\mathcal{V}_i^A$.

The discussion above not only demonstrates that a natural and direct connection is maintained between the gadgets and the underlying model-checking problem, but also that the gadgets themselves can be a site for the following form of information abstraction. Instead of including in a gadget H_i all the relevant summary functions, one can include only summary functions that assign the value $\dashv$ to a given subset of the exits. Note that in this case we must also add a move from t^{must} and t^{may} to the special node $\dashv$ of the sub-arena, to allow Player 0 to force a tie in case he is not happy with this limited choice of summary functions. By considering only a subset of the summary functions one can drastically reduce the number of nodes in the gadget. In fact, we believe that in many cases one can consider summary functions that assign $\dashv$ to almost all the exits. The reason for this optimism is that the hierarchical structure of the system usually reflects a corresponding hierarchical division of responsibility. Thus, in many cases, certain sub-structures will be responsible for satisfying certain parts of the specification. Thus, exits of the form (e, ψ'), where the sub-formula ψ' should by design be satisfied inside the sub-structure that the gadget represents (or that have a disjunctive alternative that should be satisfied inside the sub-structure), should be assigned the value $\dashv$. It is interesting to note that if the formula fails to validate when considering only summary functions that assign $\dashv$ to such exits, but does validate when considering all summary functions, then there is a bug in the sense that the designer's beliefs about the division of work between the different sub-structures in the system is wrong.

References

[1] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T.W. Reps, M. Yannakakis, Analysis of recursive state machines, ACM Trans. Program. Lang. Syst. 27 (4) (2005) 786–818.

[2] R. Alur, S. Chaudhuri, K. Etessami, P. Madhusudan, On-the-fly reachability and cycle detection for recursive state machines, in: Proc. 11th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, in: Lecture Notes in Comput. Sci., vol. 3440, Springer, 2005, pp. 61–76.

[3] R. Alur, S. Chaudhuri, P. Madhusudan, Languages of nested trees, in: Proc. 18th Int. Conf. on Computer Aided Verification, in: Lecture Notes in Comput. Sci., vol. 4144, Springer, 2006, pp. 329–342.

[4] R. Alur, K. Etessami, M. Yannakakis, Analysis of recursive state machines, in: Proc. 13th Int. Conf. on Computer Aided Verification, in: Lecture Notes in Comput. Sci., vol. 2102, Springer, 2001, pp. 207–220.

[5] R. Alur, S. Kannan, M. Yannakakis, Communicating hierarchical state machines, in: Proc. 26th Int. Colloq. on Automata, Languages, and Programming, in: Lecture Notes in Comput. Sci., vol. 1644, Springer, 1999, pp. 169–178.

[6] R. Alur, M. Yannakakis, Model checking of hierarchical state machines, ACM Trans. Program. Lang. Syst. 23 (3) (2001) 273–303.

[7] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: Application to model-checking, in: Proc. 8th Conf. on Concurrency Theory, in: Lecture Notes in Comput. Sci., vol. 1243, Springer, 1997, pp. 135–150.

[8] L. Bozzelli, Complexity results on branching-time pushdown model checking, Theoret. Comput. Sci. 379 (1–2) (2007) 286–297.

[9] E. Clarke, E. Emerson, A. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Trans. Program. Lang. Syst. 8 (2) (1986) 244–263.

[10] E. Clarke, O. Grumberg, D. Peled, Model Checking, MIT Press, 1999.

[11] R. Cleaveland, A linear-time model-checking algorithm for the alternation-free modal μ-calculus, Form. Methods Syst. Des. 2 (1993) 121–147.

[12] D. Dams, R. Gerth, O. Grumberg, Abstract interpretation of reactive systems, ACM Trans. Program. Lang. Syst. 19 (2) (1997) 253–291.

[13] W.-P. de Roever, H. Langmaack, A. Pnueli (Eds.), Compositionality: The Significant Difference. Proceedings of Compositionality Workshop, Lecture Notes in Comput. Sci., vol. 1536, Springer, 1998.

[14] D. Drusinsky, D. Harel, On the power of bounded concurrency I: Finite automata, J. ACM 41 (3) (1994) 517–539.

[15] E. Emerson, C. Jutla, Tree automata, μ-calculus and determinacy, in: Proc. 32nd IEEE Symp. on Foundations of Computer Science, 1991, pp. 368–377.

[16] E. Emerson, C.-L. Lei, Temporal model checking under generalized fairness constraints, in: Proc. 18th Hawaii Int. Conf. on System Sciences, Western Periodicals Company, 1985.

[17] P. Godefroid, R. Jagadeesan, Automatic abstraction using generalized model checking, in: Proc. 14th Int. Conf. on Computer Aided Verification, vol. 2404, 2002, pp. 137–150.
[18] S. Göller, M. Lohrey, Fixpoint logics on hierarchical structures, in: Proc. 25th Conf. on Foundations of Software Technology and Theoretical Computer Science, in: Lecture Notes in Comput. Sci., vol. 3821, Springer, 2005, pp. 483–494.
[19] O. Grumberg, M. Lange, M. Leucker, S. Shoham, When not losing is better than winning: Abstraction and refinement for the full mu-calculus, Inform. and Comput. 205 (8) (2007) 1130–1148.
[20] D. Harel, O. Kupferman, M. Vardi, On the complexity of verifying concurrent transition systems, Inform. and Comput. 173 (2002) 1–19.
[21] M. Huth, R. Jagadeesan, D. Schmidt, Model checking partial state spaces with 3-valued temporal logics, in: ESOP, 2001, pp. 155–169.
[22] D. Janin, I. Walukiewicz, Automata for the modal μ-calculus and related results, in: 20th Int. Symp. on Mathematical Foundations of Computer Science, in: Lecture Notes in Comput. Sci., vol. 969, Springer, 1995, pp. 552–562.
[23] O. Kupferman, M. Vardi, P. Wolper, An automata-theoretic approach to branching-time model checking, J. ACM 47 (2) (2000) 312–360.
[24] S. La Torre, M. Napoli, M. Parente, G. Parlato, Verification of scope-dependent hierarchical state machines, Inform. and Comput. 206 (9–10) (2008) 1161–1177.
[25] K. Larsen, G. Thomsen, A modal process logic, in: Proc. 3rd IEEE Symp. on Logic in Computer Science, 1988.
[26] D. Muller, P. Schupp, Alternating automata on infinite trees, Theoret. Comput. Sci. 54 (1987) 267–276.
[27] A. Murano, M. Napoli, M. Parente, Program complexity in hierarchical module checking, in: LPAR'08, in: Lecture Notes in Comput. Sci., vol. 5330, Springer, 2008, pp. 318–332.
[28] A. Pnueli, In transition from global to modular temporal reasoning about programs, in: K. Apt (Ed.), Logics and Models of Concurrent Systems, in: NATO Advanced Summer Institutes, vol. F-13, Springer, 1985, pp. 123–144.
[29] S. Qadeer, Taming concurrency: A program verification perspective, in: 19th Int. Conf. on Concurrency Theory, in: Lecture Notes in Comput. Sci., vol. 5201, Springer, 2008, p. 5.
[30] A. Rabinovich, Complexity of equivalence problems for concurrent systems of finite agents, Inform. and Comput. 139 (2) (1997) 111–129.
[31] S. Shoham, O. Grumberg, A game-based framework for CTL counterexamples and 3-valued abstraction-refinement, ACM Trans. Comput. Log. 9 (1) (2007).
[32] I. Walukiewicz, Pushdown processes: games and model checking, in: Proc. 8th Int. Conf. on Computer Aided Verification, in: Lecture Notes in Comput. Sci., vol. 1102, Springer, 1996, pp. 62–74.
[33] I. Walukiewicz, Model checking CTL properties of pushdown systems, in: Proc. 20th Conf. on Foundations of Software Technology and Theoretical Computer Science, in: Lecture Notes in Comput. Sci., vol. 1974, Springer, 2000, pp. 127–138.
[34] I. Walukiewicz, Pushdown processes: Games and model-checking, Inform. and Comput. 164 (2) (2001) 234–263.
[35] T. Wilke, CTL$^+$ is exponentially more succinct than CTL, in: Proc. 19th Conf. on Foundations of Software Technology and Theoretical Computer Science, in: Lecture Notes in Comput. Sci., vol. 1738, Springer, 1999, pp. 110–121.
[36] T. Wilke, Alternating tree automata, parity games, and modal μ-calculus, Bull. Soc. Math. Belg. 8 (2) (2001).